1. Preface
    1. Preface, Introduction to XML
2. ITSE1356
    1. A Brief Introduction to XML
    2. XML - Tags, Elements, Content, and Attributes
    3. XML - Well-Formed and Valid Documents
    4. Xml0100 Writing XML Review
    5. Xml0110 Transforming XML Review
    6. Xml0120 Validating XML Review
    7. Json0110: Preface to JSON
    8. Json0120: What is JSON?
    9. Json0130: JSON and JavaScript
    10. Json0140-Calling External JavaScript Functions
3. Flex
    1. XML - Namespaces - Flex 3
    2. XML - Namespaces - Flex 4
    3. The Default Application Container - Flex 3 and Flex 4
    4. Using Flex 3 in a Flex 4 World
    5. Handling Slider Change Events in Flex 3 and Flex 4
    6. Flex Resources

Preface, Introduction to XML
This module is a preface to the collection titled Introduction to XML.

Table of Contents

Welcome

Welcome to my collection titled *Introduction to XML* . This collection contains a set of lesson modules designed to teach the fundamentals of XML. Some of the modules use Adobe Flex as the teaching vehicle.

This collection consists of two major sub-collections:

- ITSE1356
- Flex

ITSE1356

The material in the ITSE1356 sub-collection, beginning with the module titled A Brief Introduction to XML , contains material that I use to teach the course identified as **ITSE 1356 - Extensible Markup Language (XML)** at Austin Community College in Austin, TX.

As of the Spring 2014 semester, the textbook for this course is *XML: Visual QuickStart Guide, 2nd Edition* , By Kevin Howard Goldberg.

Fundamentals of XML

The first three modules in the ITSE1356 sub-collection provide tutorial material written by me on the fundamentals of XML including:

- Structured documents

- Tags
- Elements
- Content
- Attributes
- Valid XML documents
- Well-formed XML documents

These modules are provided to supplement the material in the Goldberg textbook.

Review questions

The next three modules in the ITSE1356 sub-collection provide review questions, answers, and explanations keyed to various chapters of the Goldberg textbook.

JSON

The review modules are followed by several modules on **JSON** *(JavaScript Object Notation)* . As a lightweight data-interchange format, JSON is emerging as a strong alternative to the use of XML for that purpose.

An introductory section on JSON is scheduled to be introduced into the ITSE 1356 course beginning in the Fall 2014 semester. JSON material is not included in the Goldberg textbook mentioned earlier. Therefore, these JSON modules will constitute the primary learning resource for the JSON section of the course.

Flex

In addition to being the material that I use to teach the course identified above, knowledge of the material in this sub-collection is a prerequisite for understanding the material in the Flex sub-collection discussed below.

Flex

The Flex material is not a part of the ITSE1356 course.

Flex is an XML application that can be used to create programs that run in Adobe's Flash player. Flex is an alternative to the approach explained in Object-Oriented Programming (OOP) with ActionScript .

The material in the ITSE1356 sub-collection is somewhat theoretical in nature. The material in the Flex sub-collection, beginning with the module titled XML - Namespaces - Flex 3 , explains in practical terms how to use the Flex application.

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Preface
- File: Flex0070Preface.htm
- Published: 11/08/13
- Revised: 02/02/14

on cnx.org and that it was made and published without my prior knowledge.
Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

A Brief Introduction to XML
This module is part of a collection dedicated to learning XML.

Table of Contents

Preface

General

This module is part of a collection dedicated to learning XML.

Viewing tip

I recommend that you open another copy of this document in a separate browser window and use the following links to easily find and view the figures while you are reading about them.

Figures

Supplemental material

I recommend that you also study the other lessons in my extensive collection of online programming tutorials. You will find a consolidated index at www.DickBaldwin.com .

A brief introduction to XML

The name **XML** derives from e **X** tensible **M** arkup **L** anguage. According to Wikipedia,

> *"A markup language is a system for annotating text in a way which is syntactically distinguishable from that text."*

In other words, when text has been annotated or marked up, the annotations can be easily distinguished from the original text. For example, if you turn in a term paper and the professor annotates it with a red pencil, you can easily distinguish her annotations from your original text. However, XML doesn't use color to annotate text. Instead, XML uses specially formatted text to annotate text.

Structured documents

XML gives us a way to create and maintain structured documents in plain text that can be rendered in a variety of different ways. For example, before I upload this document to the Connexions website for publishing, I will convert into **CNXML** , which is one of the many flavors of XML. Once the document is on the website in that format, programs on the website have the ability to render it in the form of a web page *(which you are probably reading right now)* or in the form of a PDF document, which you can download and print if you choose to do so.

There is a lot of jargon involved in XML. One of my objectives will be to explain the jargon.

What do I mean by a "structured document?"

I will answer this question by providing an example. A book is a structured document. In its simplest form, a book may be composed of chapters. The chapters may be composed of sections. The sections may contain illustrations and tables. The tables are composed of rows and columns. Thus, it would be possible to draw a picture that illustrates the structure of a book.

What do I mean by "plain text?"

Characters such as the letters of the alphabet and punctuation marks are represented in the computer by numeric values, similar to a simple substitution code that a child might devise. For example in one popular encoding scheme (ASCII), the upper-case version of the character "A" is represented by the value 65, a "B" is represented by the value 66, a "C" is represented by 67, etc.

Different encoding schemes

The actual correspondence between the characters and the specific numeric values representing the characters has been described by several different encoding schemes over the years. One of the most common and enduring schemes is a scheme that was devised a number of years ago by an organization known as the American Standards Committee on Information Interchange. This encoding scheme is commonly known as the ASCII code.

XML supports several encoding schemes

XML is not confined to the use of the ASCII encoding scheme. Several different encoding schemes can be used. However, all of them have been selected to make it possible to read a raw XML document without the requirement for any special software.

What do I mean by a raw XML document?

By a raw XML document, I am referring to the string of sequential characters that makes up the document, before any specific rendering has

been applied to the document.

What do I mean by rendering?

The most common use of the word rendering in the information technology world means to present something for human consumption. Thus, we render the specifications for a new house as a set of drawings.

When we speak of rendering a drawing or an image, we usually mean that we are going to present it in a way that makes it look like a drawing or an image to a human observer.

When we speak of rendering a document, we usually mean that we are going to present it in a way that a human will recognize such as a book, a newspaper, a web page, or some other document style.

Consider a newspaper, for example

There are at least two different ways to render a newspaper. One way is to print the information *(daily news)* , mostly in black and white, on large sheets of low-grade paper commonly known as newsprint. This is the rendering format that ends up on my driveway each morning.

Render on a computer screen

Another way to render a newspaper is to present the information on a computer screen, usually in full color, with the information content trying to fight its way through dozens of animated advertisements. This is the rendering format that ends up on my computer screen each day when I check for the news of the day.

The base information doesn't change

The base information for the newspaper doesn't *(or shouldn't)* change for these two renderings. After all, news is news and the content of the news shouldn't depend on how it is presented. What does change is the manner in which that information is presented.

A newspaper is a structured document

A newspaper is a structured document consisting of pages, columns, etc. When the information content of a newspaper is created and maintained in XML, that same information content can be rendered either on newsprint paper or on your computer screen without having to rewrite the information content.

Achieving Structure

Consider the simple structure shown in Figure 1 that represents a book having two chapters with some text in each chapter:

Figure 1 . The structure of a simple book.

```
Begin Book

  Begin Chapter 1
    Text for Chapter 1
  End Chapter 1

  Begin Chapter 2
    Text for Chapter 2
  End Chapter 2

End Book
```

Obviously a real book has a lot more structure than this, such as the preface, the table of contents, paragraphs in the text, and an alphabetical index. However, I am trying to keep this example as simple as possible.

The Objective of XML

Perhaps the primary reason for using XML is to make it possible to share the same physical document among different computer systems in a way that they all understand.

No small task

That is no small task. Over the years, dozens of different types of computers have been built, operating under several different operating systems, and running thousands of different programs. As a result, insofar as the exchange of structured documents is concerned, the computer world is a modern manifestation of the "Tower of Babel" where everyone spoke a different language. XML attempts to rectify this situation by providing a common language for structured documents.

What Does XML Contribute?

Without getting into the technical details at this point, XML provides a definition of a simple scheme by which the structure and the content of a document can be established. The resulting physical document is so simple that any computer *(or any human)* can read it with only a modest amount of preparation. You will sometimes see XML referred to as a "meta" language.

What Does Meta Mean?

In computer jargon, the term meta is often used to identify something that provides information about something else. *(If you want to impress someone at your next cocktail party, mention that meta information is information about information.)*

For example, consider the listings of stock prices, bond prices, and mutual fund prices that commonly appear in most daily newspapers. The various tables on the page provide information about the bid and ask prices for the various stock, bond, and mutual fund instruments.

Usually somewhere on the page, you will find an explanation as to how to interpret the information presented throughout the remainder of the page. You could probably think of the information contained in the explanation as meta information. It provides information about other information.

So, why might people refer to XML as a meta language?

If you write a book, XML doesn't tell you how to structure the document that represents your book. Rather, it provides you with a set of rules that you can use to establish structure and content when you create the document that represents your book. It is up to you to decide how you will use those rules to establish the structure and content of your book.

Information about new languages

You might say that XML is a language that provides information about a new language that you are free to invent. For example, Flex is a specialized programming language that is based on XML. XML doesn't specify the language. Instead, XML provides the tools used by the inventors of the Flex programming language to specify the structure of the language.

Different flavors of XML

Similarly, XML doesn't specify CNXML. Instead, XML provides the tools used by the inventors of CNXML to specify the format of documents suitable for publication on the Connexions website. In the past, I have also published documents on a particular IBM website. That website uses a different flavor of XML to specify the format of documents suitable for publication on the website.

Transportable

If you follow the rules for creating an XML document, the document that you create can be easily transported among various computers and rendered in a variety of different ways.

Multiple renderings

For example, you might want to have two different renderings of your book. One rendering might be in conventional printed format and the other rendering might be in an online format. The use of XML makes it practical to render your book in two or more different ways without any requirement to modify the original document that you produce.

Applying XML

At this point, I am going to provide two different examples of actual XML code, either of which might reasonably represent the simple book example presented earlier in Figure 1. The first example is shown in Figure 2.

Figure 2 . Very simple XML syntax.

```
<book>
  <chap>
    Text for Chapter 1
  </chap>

  <chap>
    Text for Chapter 2
  </chap>
</book>
```

If you compare this example with the book example given earlier , you should be able to see a one-to-one correspondence between the "elements" in this XML code and the description of the book presented earlier.

Introducing attributes

The example in Figure 3 provides an improvement over the example in Figure 2. Figure 3 provides an "attribute" in each of the chapter elements. Each attribute specifies the chapter number.

Figure 3 . XML syntax with attributes.

```
<book>
  <chap number="1">
    Text for Chapter 1
  </chap>

  <chap number="2">
    Text for Chapter 2
  </chap>
</book>
```

That's a wrap

That's enough for this module. In the next module, I will begin discussing the syntax shown in Figure 3 and begin the explanation of *tags* , *elements* , *content* , and *attributes* .

Miscellaneous

This section contains a variety of miscellaneous materials.

Note: Housekeeping material

- Module name: A Brief Introduction to XML
- File: FlexXhtml0080.htm
- Revised: 12/02/14

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.
I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module.
In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.
Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

XML - Tags, Elements, Content, and Attributes
This module is part of a collection dedicated to learning XML.

Table of Contents

Preface

General

This tutorial lesson is part of a series dedicated to learning XML.

Viewing tip

I recommend that you open another copy of this document in a separate browser window and use the following links to easily find and view the figures and listings while you are reading about them.

Figures

Listings

Supplemental material

I recommend that you also study the other lessons in my extensive collection of online programming tutorials. You will find a consolidated index at www.DickBaldwin.com .

Tags, elements, content, and attributes

XML can be used to produce a variety of applications. Flex is an XML application developed by Adobe that can be used to produce code that will run in the Adobe Flash player.

Listing 1 shows the code from a very simple Flex MXML file. *(Note that the code shown in Listing 1 is from Flex version 3.x. Code from version 4.x would be different in several respects.)*

Listing 1 . Simple Flex MXML code.

Listing 1 . Simple Flex MXML code.

```
<?xml version="1.0" encoding="utf-8"?>
<!--DragAndDrop04-->

<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:cc="CustomClasses.*">
  <cc:Driver/>

</mx:Application>
```

I'm not going to explain the MXML code in this lesson. I am simply providing the code as an example of an XML document so you can see what a real XML document looks like.

What is a tag?

I will refer to items enclosed in angle brackets, such as those shown in Figure 1, as tags.

Figure 1 . Tag example.

Figure 1 . Tag example.

```
<book>

...

</book>
```

The first tag shown in Figure 1 is often referred to as a *start tag* or an *opening tag* . The second tag is often referred to as an *end tag* or a *closing tag* .

Note that the start tag and the end tag differ only in that the end tag contains a slash character. However, the start tag can also contain optional namespace indicators and attributes as discussed below.

What are elements, content, and attributes?

Listing 2 contains a *start tag* and an *end tag* with an *attribute* and some *content* .

Listing 2 . An element containing tags, content, and an attribute.

```
<chap number="1">
  Text for Chapter 1
</chap>
```

An element

The entire set of characters beginning with the start tag and ending with the end tag constitutes an element .

An element usually consists of a start tag and an end tag with the content sandwiched in between the two tags, but there are exceptions to that rule. You will learn about those exceptions, including empty elements later.

The tags

You have probably already recognized the tags in Listing 2 as the two sets of characters beginning with a left angle bracket and ending with a right angle bracket.

The start tag may contain optional attributes. In Listing 2, a single attribute provides the *number* value for the chapter.

The start tag may also contain *namespace* information. There is no namespace information in Listing 2. You will learn about namespaces in a future lesson.

The content

The set of characters in between the tags constitutes the *content* .

An attribute

The set of characters following the word chap in the start tag constitutes an *attribute* .

The term attribute is commonly used in computer science and usually has about the same meaning, regardless of whether the discussion revolves around XML, Java programming, or database management. An attribute often serves to partially describe the thing to which it refers.

Things have attributes

A chapter in a book is a thing and a chapter has attributes such as its number. A person is also a thing. Therefore, a person also has attributes. Each attribute has a value. Here is a list of some of the attributes *(along with their values)* that might be used to describe a person:

- name="Joe"
- height="84"
- weight="176"
- complexion="pale"
- sex="male"
- training="Java programmer"
- degree="Masters"

Obviously, there are many more attributes that could be used to describe a person.

The importance of an attribute depends on the context

Whether or not a particular attribute for a person is important depends on the context in which the person is being considered. For example, if the person is being considered in the context of a candidate for a basketball team, the height, weight, and sex attributes will probably be important.

On the other hand, if the person is being considered in the context of a candidate for employment as a computer programmer, the height, weight, and sex attributes should not be important at all, but the training and degree attributes might be very important.

Why does XML use attributes?

The description of XML that I provided in an earlier lesson is repeated here for convenience:

> "XML gives us a way to create and maintain structured documents in plain text that can be rendered in a variety of different ways."

Attributes often provide information that is needed for the rendering process, but attributes have many other uses as well.

Rendering

I suggested earlier that the most common modern use of the word rendering means to present something for human consumption. I gave an example of a newspaper that can either be rendered on newsprint paper or can be rendered on a computer screen.

A rendering engine

If the newspaper *(structured document)* is created and maintained as an XML document, then some sort of computer program *(often referred to as a rendering engine)* will probably be used to render it into the desired presentation format.

This document

For example, the original version of this document was created as a special flavor of an XML document known as an XHTML document. Listing 3 shows a short sample of the raw XML code in the original document.

Listing 3 . A section of raw XHTML code.

Listing 3 . A section of raw XHTML code.

```
 <p> <strong style="color:#ff0000" ">A
rendering
engine  "</strong> </p>
<p>If the newspaper (structured document) is
created and maintained as an XML document,
then
some sort of computer program (often referred
to
as a rendering engine) will probably be used
to
render it into the desired presentation
format. </p>
 <p>For example, this document is a special
flavor
of an XML document known as an XHTML document.
Listing 3 shows a short sample of the raw XML
code that was delivered to your computer from
my
website.  </p>
```

Created using a WYSIWYG XHTML editor

I originally created the document as an XHTML document using a WYSIWYG XHTML editor that behaves much like a word processor. *(If you don't know what WYSIWYG means, Google it.)* Of course, the document has since undergone quite a lot of editing so the final XHTML version probably doesn't match the XHTML code in Listing 3.

Transform to CNXML

Later on, I used a Java program of my own design to transform the final XHTML document into another flavor of XML known as CNXML for publishing on the Connexions website.

That illustrates another characteristic of XML. Because the formats of certain flavors of XML documents are well defined, it is often practical to transform them from one flavor to another flavor.

That makes it possible for me to create the document using a program that is very similar to a word processor and then transform the output of that program into a fairly cryptic format that satisfies the publishing requirements of the website.

Your browser is rendering the document

When you accessed the document from the Connexions website, it was transformed back into an XHTML document and sent to your computer.

As you can see in Listing 3, viewing raw XHTML isn't very enjoyable. Fortunately, your browser is acting as a rendering engine to render the raw XHTML text into a much more pleasing presentation format.

Back to the book example

A book that is created and maintained as an XML document could be rendered in a variety of different ways. Whichever way it is rendered, however, it would probably be useful to separate and number the chapters. Therefore, the value of the *number* attribute could be used by the rendering engine to present the chapter number for a specific rendering.

In some renderings, the number might appear on an otherwise blank page that begins a new chapter. In a different rendering, the chapter number might appear in the upper right or left-hand corner of each page.

Tell me again, what is an element?

As I explained earlier, an *element* usually consists of a *start tag (with optional attributes and namespace information)* , an *end tag* , and the

content sandwiched in between as shown earlier in Listing 2.

Elements can be nested

Elements can be nested inside other elements as shown in Listing 4.

Listing 4 . Nested elements.

```
<book>
  <chapter number="1">
    Content for Chapter 1
  </chapter>

  <chapter number="2">
    Content for Chapter 2
  </chapter>
</book>
```

In Listing 4, two chapter elements are nested inside a book element.

Why does XML use elements?

It is probably fair to say that the element constitutes the fundamental unit of information in an XML document. For example, the element defines the type of information, such as chapter in our book example.

Sandwiched in between the start tag and the end tag of an element, we find the raw information *(content)* that the XML document is designed to convey. For a text document, you are likely to find a lot of content between the tags. For example, in Listing 3, there are several lines of text between the paragraph tags identified by the p and the /p enclosed in angle brackets.

Once again, what is content?

Of the four terms mentioned earlier, *(tags, elements, content, and attributes)* , content is the easy one. Content is sandwiched in between the start tag and the end tag of an element. Usually the content of the elements contains the information that the XML document is designed to convey. In other words, this is where we put the information for which the document was created. The tags and attributes are there to create the structure.

For example, if the XML document is being used for the creation and maintenance of material for a newspaper, the content constitutes the news. If the XML document is being used for the creation and maintenance of a Java programming textbook, the content contains the information about Java programming that we want to convey to the student.

Why do we need structure?

One of the primary objectives of XML is to separate content from presentation. If we insert the raw material as content into a structure defined by the tags, elements, and attributes, then that raw material can be presented in a variety of ways.

Miscellaneous

This section contains a variety of miscellaneous materials.

Note: Housekeeping material

- Module name: XML - Tags, Elements, Content, and Attributes
- File: FlexXhtml0082.htm
- Revised: 12/02/14

Note: Disclaimers:

Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.
I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module.
In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.
Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

XML - Well-Formed and Valid Documents
This module is part of a collection dedicated to learning XML.

Table of Contents

Preface

General

This module is part of a collection dedicated to learning XML.

Viewing tip

I recommend that you open another copy of this document in a separate browser window and use the following links to easily find and view the figures and listings while you are reading about them.

Supplemental material

I recommend that you also study the other lessons in my extensive collection of online programming tutorials. You will find a consolidated index at www.DickBaldwin.com .

Well-formed and valid documents

In previous lessons, I have discussed tags, elements, content, and attributes in detail. The time has come to take up the following topics:

- Well-formed documents
- Valid documents
- The DTD

Valid documents and the DTD

What is a DTD?

Figure 1 contains a quotation from the XML FAQ that describes a DTD.

Figure 1 . What is a DTD?

```
"A DTD is usually a file (or several files to
be used
together) which contains a formal definition
of a particular
type of document. This sets out what names can
be used for
elements, where they may occur, and how they
all fit together.
For example, if you want a document type to
describe <LIST>s
which contain <ITEM>s, part of your DTD would
contain
something like

<!ELEMENT item (#pcdata)>
<!ELEMENT list (item)+>

This defines items containing text, and lists
containing
items.

It's a formal language which lets processors
automatically
parse a document and identify where every
element comes and
how they relate to each other, so that
stylesheets, navigators,
browsers, search engines, databases, printing
routines, and
other applications can be used."
```

DTDs are complicated

I included the above quotation to emphasize one very important point -- DTDs are complicated. The creation of a DTD of any significance is a very complex task.

The good news!

The good news is that many of you will never need to worry about having to create a DTD for two reasons:

1. In the most fundamental sense, XML does not require the use of a DTD.
2. Even when it is advisable to use a DTD with XML, someone else may already have created the DTD on your behalf.

A validating XHTML editor

For example, I wrote the original version of this HTML document using a validating XHTML editor named Amaya . Even though the editor uses a DTD to confirm that my document is a valid XHTML document *(and warns me if it isn't)* , it wasn't necessary for me to write the DTD. The people who wrote the editor also wrote the DTD.

Three Parts

It is reasonable to think of an XML document as consisting of three parts, some of which are optional. I'm going to refer to the parts as files just so I will have something to call them *(but they don't have to be separate physical files)* .

One file contains the information content of the document *(words, pictures, etc.)* . This is the part containing tags, elements, content, and attributes that the author wants to expose to the client. I have discussed this part in previous lessons.

A second file is the DTD, which meets the definition given above.

A third file is a stylesheet that establishes how the content that conforms to the DTD is to be rendered on the output device. This is how the author

wants the material to be presented to the client.

Rendering

For example a tag with an attribute of "red" might cause something to be presented bright red according to one stylesheet and dull red according to another stylesheet. *(It might even be presented as some shade of green according to still another stylesheet.)*

With XML, the DTD is optional but the stylesheet (or some processing mechanism that substitutes for a stylesheet) is generally required. Something has to be able to render the content in the manner that the author intended it to be rendered. Otherwise, the client will be forced to view the document as raw XML text, which usually isn't very enjoyable.

A DTD can be very complex

Once again, according to the **XML FAQ** :

Note:
"... the design and construction of a DTD can be a complex and non-trivial task, so XML has been designed so it can be used either with or without a DTD. DTDless operation means you can invent markup without having to define it formally. To make this work, a DTDless file in effect 'defines' its own markup, informally, by the existence and location of elements where you create them. But when an XML application such as a browser encounters a DTDless file, it needs to be able to understand the document structure as it reads it, because it has no DTD to tell it what to expect, so some changes have been made to the rules."

Without the technical jargon please

In other words, it is entirely possible to create an XML document without the requirement for a DTD.

What is a valid document?

In the normal sense of the word, if something is *invalid* , that usually means that it is not any good. However, that is not the case for XML. An invalid XML document can be a perfectly good and useful document.

A valid XML document is one that conforms to an existing DTD in every respect.

In other words, unless the DTD allows a tag with the name "color", an XML document being validated against that DTD containing a tag with that name is not valid.

However, because XML does not require a DTD, an XML processor cannot require validation of the document. Many very useful XML documents are not valid, simply because they were not constructed according to an existing DTD.

An XHTML document

The document that you are now reading was originally created as a valid XML document before being transformed to CNXML and uploaded to the Connexions website. It was created as a special flavor of XML known as XHTML. As I mentioned earlier, the document was created using W3C's WYSIWYG Editor/Browser named Amaya . *(Subsequent edited versions have been created using Microsoft Expression Web versions 3 and 4.)*

What you are probably reading now is a rendered version of the document after having gone through a couple of edits and transformations. However, if you were to have looked at the raw XHTML code at the beginning of the document before it was transformed to CNXML, you would have seen something like the XML code shown in Listing 1.

Listing 1 . Raw XHTML code.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="content-type"
content="text/html; charset=iso-8859-1" />
<title>Flex Programming by Richard G.
Baldwin</title>
<meta name="generator"
content="Amaya, see http://www.w3.org/Amaya/"
/>
</head>
```

(Note that some extra line breaks were inserted in Listing 1 to force it to fit into this narrow publication format.)

The DTD

Note in particular the code that begins with "http: in Listing 1. This code specifies the DTD that is used to validate the XML code. If I had inadvertently entered some XML code that caused the document to become invalid, a red warning would have appeared in the bottom right corner of the Amaya editor.

A download site

If you examine the DTD information in Listing 1 carefully, you will see that it actually specifies a location on the Internet from which you can download the DTD file. You can download it and open it in a text editor, such as Windows Notepad, to see a sample of a really complicated DTD.

Listing 2 shows a small portion of the XHTML DTD downloaded from the address shown in Listing 1.

Listing 2 . A small portion of the XHTML DTD.

```
<!--
   Extensible HTML version 1.0 Transitional
DTD

   This is the same as HTML 4 Transitional
except for
   changes due to the differences between XML
and SGML.

   Namespace = http://www.w3.org/1999/xhtml

   For further information, see:
http://www.w3.org/TR/
xhtml1

   Copyright (c) 1998-2002 W3C (MIT, INRIA,
Keio),
   All Rights Reserved.

   This DTD module is identified by the PUBLIC
and
SYSTEM identifiers:

   PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
   SYSTEM
```

Listing 2 . A small portion of the XHTML DTD.

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd"

   $Revision: 1.2 $
   $Date: 2009-12-14$

-->

<!--====== Character mnemonic entities
=============-->

<!ENTITY % HTMLlat1 PUBLIC
   "-//W3C//ENTITIES Latin 1 for XHTML//EN"
   "xhtml-lat1.ent">
%HTMLlat1;

<!ENTITY % HTMLsymbol PUBLIC
   "-//W3C//ENTITIES Symbols for XHTML//EN"
   "xhtml-symbol.ent">
%HTMLsymbol;

<!ENTITY % HTMLspecial PUBLIC
   "-//W3C//ENTITIES Special for XHTML//EN"
   "xhtml-special.ent">
%HTMLspecial;
```

(Once again, I inserted some line breaks into the text in Listing 2 to force it to fit into this publication format.)

Well-formed documents

XML derives from an earlier more complicated markup language known as SGML. Being well-formed is not a property of SGML. The concept of being well-formed was introduced as a requirement of XML, apparently to deal with the situation where a DTD is not available.

Why do we need well-formed XML documents?

Once again, according to the XML FAQ :

Figure 2 . Why do we need well-formed XML documents?

```
"For example, HTML's <img> element is defined
as `EMPTY':
it doesn't have an end-tag. Without a DTD, an
XML application
would have no way to know whether or not to
expect an end-tag
for an element, so the concept of `well-
formed' has been
introduced.

This makes the start and end of every element,
and the
occurrence of EMPTY elements completely
unambiguous."
```

All XML documents must be well-formed

XML documents need not be valid, but ***ALL XML DOCUMENTS MUST BE WELL-FORMED*** .

To be well-formed...

A well-formed XML document must meet several different criteria.

To begin with, in a well-formed XML document, all elements that can contain character data must have both start and end tags.

What is character data?

For purposes of this explanation, let's just say that the content that we discussed earlier comprises character data.

Attribute values must be in quotes

All attribute values must be in quotes *(apostrophes or double quotes)* . You can surround the value with apostrophes *(single quotes)* if the attribute value contains a double quote. An attribute value that is surrounded by double quotes can contain apostrophes.

Dealing with empty elements

EMPTY elements *(those that contain no character data)* must be written in one of the two ways shown in Listing 3, and for several reasons, the first way is usually considered preferable.

Listing 3 . Required syntax for an empty element.

```
<mx:Button label="My button."/>
<mx:Button label="My button."></mx:Button>
```

Don't forget that even an EMPTY element can contain one or more attributes along with namespace information inside the start tag. *(In the case of Listing 3, **mx:** is namespace information and the **label** information is an attribute.)*

Markup characters and entities

There are also rules regarding the inclusion of markup characters.

Note:
No markup characters are allowed

```
For a document to be well-formed, it must not have
markup
characters such as angle brackets or ampersands in
the text
data. If such characters are needed, you can
represent them
using &lt; and &amp; instead.

These special combinations of characters that
represent other
characters, such as &lt; that represents the left
angle
bracket are called entities.
```

Nesting

Elements must nest properly. If one element contains another element, the entire second element must be defined inside the start and end tags of the first element. Every element in an XML document, other than the root element, is nested inside another element.

Validity and well-formed requirements recap

Valid XML files are those that have a DTD and that conform to the DTD.

All XML files must be well-formed, but there is no requirement for them to be valid.

A DTD is not required in which case validity is impossible to establish. However, if XML documents do have a DTD, they must conform to it, which makes them valid.

Why use a DTD if it is not required?

There are many reasons to use a DTD, in spite of the fact that XML doesn't require one. One reason is that the use of a DTD makes it possible to enforce format specifications. For example, in a document that represents a book, the DTD could require that paragraph elements can occur only inside of page elements. It could also require that page elements can occur only inside chapter elements. It could require that there be a preface element and that it must occur before any chapter elements.

Enforcing format specifications

For example, by creating this document using Amaya and the DTD for XHTML, I was required to produce a document that conformed to the DTD for XHTML documents. Otherwise, I would have gotten warnings from the editor and would have been required to acknowledge that the document didn't conform to the DTD in order to save it.

On one hand, that sounds like a lot of hassle. On the other hand, by creating a document that conforms to the DTD for XHTML, I can be sure that it will render properly in any browser that is guaranteed to properly render XHTML documents.

Miscellaneous

This section contains a variety of miscellaneous materials.

Note: Housekeeping material

- Module name: XML - Well-Formed and Valid Documents
- File: FlexXhtml0084.htm
- Revised: 08/17/15

Note: Disclaimers:
Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.
I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.
Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Xml0100 Writing XML Review
This module contains review questions, answers, and explanations keyed to the Introduction and Chapter 1 of the textbook titled XML: Visual QuickStart Guide, 2nd Edition. By Kevin Howard Goldberg.

Table of Contents

Preface

This material is published in support of the course identified as ITSE 1356 - Extensible Markup Language (XML) at Austin Community College in Austin, TX. As of the Spring 2014 semester, the textbook for this course is XML: Visual QuickStart Guide, 2nd Edition. By Kevin Howard Goldberg.

This module contains review questions, answers, and explanations keyed to the Introduction and Chapter 1 of the textbook.

The questions and the answers are connected by hyperlinks to make it easy for you to navigate from the question to the answer and back again.

Questions

Question 1 .

True or False: XML is an abbreviation for "eXtending More Leverage."

Answer 1

Question 2

True or False: XML is a specification for storing information.

Answer 2

Question 3

True or False: XML is a specification for describing the structure of information.

Answer 3

Question 4

True or False: XML has a large vocabulary of predefined tags.

Answer 4

Question 5

True or False: XML is a markup language.

Answer 5

Question 6

True or False: XML is a set of rules for defining custom markup languages.

Answer 6

Question 7

True or False: The XML specification makes it possible for people to define their own markup languages so that others cannot use those markup languages.

Answer 7

Question 8

True or False: HTML is a language for storing and carrying information while XML is a language for displaying information.

Answer 8

Question 9

True or False: XML is easily extended and adapted.

Answer 9

Question 10

True or False: While XML works well with computers, it was not designed to be easily read by humans.

Answer 10

Question 11

True or False: XML is a proprietary specification that was invented by Microsoft.

Answer 11

Question 12

True or False: While HTML is used to format data for display, XML describes and is the data.

Answer 12

Question 13

True or False: All modern browsers know how to display an XML document in the manner intended by the author of the document.

Answer 13

Question 14

True or False: One way to specify how an XML document is to be displayed is by using XSL or eXtensible Stylesheet Language.

Answer 14

Question 15

True or False: XSL is made up of three languages: XSLT, XPath, and XML.

Answer 15

Question 16

True or False: Because every author is free to define her own tags, it is not possible to define the structure of an XML document.

Answer 16

Question 17

True or False: All standard browsers can read XML documents, use XML schemas *(DTD and XML Schema)*, and interpret XSL to format and display XML documents.

Answer 17

Question 18

True or False: It is common practice to use XML to manage and organize information and to use XSL to convert the XML into HTML.

Answer 18

Question 19

True or False: XML is a language.

Answer 19

Question 20

True or False: Every custom markup language created using the XML specification must adhere to XML's underlying grammar.

Answer 20

Question 21

True or False: XML documents are saved with a .rss extension.

Answer 21

Question 22

True or False: An XML document simply stores and describes data.

Answer 22

Question 23

True or False: Good practice suggests that the names of the tags in an XML document should describe the data they contain.

Answer 23

Question 24

True or False: The first line of an XML document, *(known as the XML declaration)* , should be as shown in Listing 1 below *(although the version number may be different at some point in the future):*

Listing 1 . Question 24.

```
<?xml version="1.0"?>
```

Answer 24

Question 25

True or False: Just like HTML documents, XML documents may have multiple root elements.

Answer 25

Question 26

True or False: A root element can have one or more child elements that describe the root element in more detail.

Answer 26

Question 27

True or False: In Listing 2 below, **units** is called a parameter

Listing 2 . Question 27.

```
<height units="inches">66</height>
```

Answer 27

Question 28

True or False: The terminology "well-formed" is commonly used in conjunction with an XML document.

Answer 28

Question 29

True or False: All of the following must be true in order for an XML document to be well-formed:

- A root element is required.
- Closing tags are required.
- Elements must be properly nested.
- Values must be enclosed in quotation marks.
- The document must be valid.

Answer 29

Question 30

True or False: An XML attribute is the most basic unit of an XML document.

Answer 30

Question 31

True or False: An XML element can contain text, attributes, and other elements.

Answer 31

Question 32

True or False: Attributes are contained within an element's opening tag and have values that are delimited by quotation marks.

Answer 32

Question 33

True or False: Tags that begin with the characters in the first line of Listing 3 and end with the characters in the second line of Listing 3 are called processing instructions.

Listing 3 . Question 33.

Listing 3 . Question 33.

```
<?
?>
```

Answer 33

Question 34

True or False: Processing instructions must be enclosed by the opening and closing tags of the root element.

Answer 34

Question 35

True or False: XML is not sensitive to upper and lower case.

Answer 35

Question 36

True or False: Element and attribute names must begin with a letter, a number, an underscore, or a colon.

Answer 36

Question 37

True or False: Closing tags are not optional in XML.

Answer 37

Question 38

True or False: Element names may begin with upper-case XML.

Answer 38

Question 39

True or False: The root element can have child elements, and child elements can have other child elements provided that they are properly nested.

Answer 39

Question 40

True or False: An attribute stores additional information about an element without adding text to the element's content.

Answer 40

Question 41

True or False: Attribute values must be in double quotes.

Answer 41

Question 42

True or False: Empty elements are elements that don't have any content of their own.

Answer 42

Question 43

True or False: Empty elements may, and frequently will have attributes that store data about the element itself.

Answer 43

Question 44

True or False: The syntax shown in Listing 4 below is the correct syntax for an empty element:

Listing 4 . Question 44.

```
<image file="my_picture.jpg" width="400"
height="200"/>
```

Answer 44

Question 45

True or False: The syntax shown in Listing 5 below is the correct syntax for an empty element:

Listing 5 . Question 45.

```
<image file="my_picture.jpg"></image>
```

Answer 45

Question 46

True or False: The correct syntax for an XML comment is shown in Listing 6 below:

Listing 6 . Question 46.

```
<?This is a comment?>
```

Answer 46

Question 47

True or False: Comments may be nested.

Answer 47

Question 48

True or False: XML has the five predefined entities shown in Listing 7 below:

Listing 7 . Question 48.

```
&amp; represents an ampersand
&lt; represents a left angle bracket
&gt; represents a right angle bracket
&quot; represents a double quotation mark
&apos; represents a single quotation mark or
apostrophe
```

Answer 48

Question 49

True or False: The CDATA syntax shown in Listing 8 below can be used to cause the XML processor to ignore angle brackets, ampersands, and other characters that might otherwise be interpreted as XML control characters:

Listing 8 . Question 49.

```
<![CDATA
5 is < 6
& is called an ampersand
]]>
```

Answer 49

Question 50

True or False: CDATA sections can be nested.

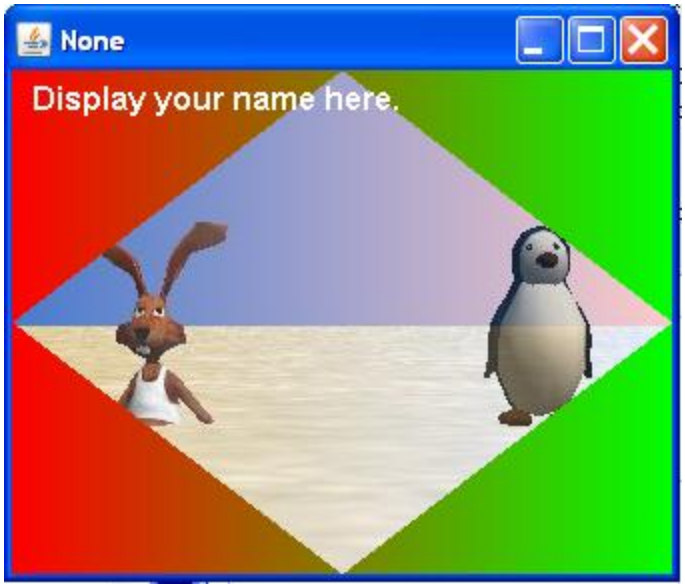
Answer 50

Listings

- Listing 9 . Answer 46..

What is the meaning of the following two images?

These images were inserted here simply to insert some space between the questions and the answers to keep them from being visible on the screen at the same time.



This image was also inserted for the purpose of inserting space between the questions and the answers.

Answers

Answer 50

False

Explanation: Goldberg page 15

Back to Question 50

Answer 49

False

There is a syntax error in the xml code. The word CDATA should be followed by a left square bracket as in ...[CDATA[

Otherwise the answer would be True because that is the purpose of the CDATA syntax.

Explanation: Goldberg page 15

Back to Question 49

Answer 48

True

Explanation: Goldberg page 14

Back to Question 48

Answer 47

False

Explanation: Comments may not be nested. Goldberg page 13

Back to Question 47

Answer 46

False

Explanation: The correct syntax for a comment is shown in Listing 9 below. Goldberg page 13

Listing 9 . Answer 46.

```
<!--This is a comment-->
```

Back to Question 46

Answer 45

True

Explanation: This is not the only correct syntax for an empty element. Goldberg page 12

Back to Question 45

Answer 44

True

Explanation: This is not the only correct syntax for an empty element. Goldberg page 12

Back to Question 44

Answer 43

True

Explanation: Goldberg page 12

Back to Question 43

Answer 42

True

Explanation: Goldberg page 12

Back to Question 42

Answer 41

False

Explanation: Attribute values must be in quotes. They can be either single or double quotes as long as they match within a single attribute. Goldberg page 11

Back to Question 41

Answer 40

True

Explanation: Goldberg page 11

Back to Question 40

Answer 39

True

Explanation: Goldberg page 10

Back to Question 39

Answer 38

False

Explanation: Element names may not begin with the letters xml in any combination of upper and lower-case characters. Goldberg page 9

Back to Question 38

Answer 37

True

Explanation: Goldberg page 9

Back to Question 37

Answer 36

False

Explanation: : Element and attribute names must begin with a letter, an underscore, or a colon. A number is not allowed. Goldberg page 8

Back to Question 36

Answer 35

False

Explanation: Case matters. Goldberg page 8

Back to Question 35

Answer 34

False

Explanation: Goldberg page 5

Back to Question 34

Answer 33

True

Explanation: For example, the XML Declaration is a processing instruction. Processing instructions are used for other purposes as well. Goldberg page 7

Back to Question 33

Answer 32

True

Explanation: Goldberg page 6

Back to Question 32

Answer 31

True

Explanation: Goldberg page 6

Back to Question 31

Answer 30

False

Explanation: An XML element is the most basic unit of an XML document. Goldberg page 6

Back to Question 30

Answer 29

False

Explanation: It is not necessary for an XML document to be valid in order to be well-formed. Goldberg page 5

Back to Question 29

Answer 28

True

Explanation: Goldberg page 5

Back to Question 28

Answer 27

False

Explanation: The term **units** is called an attribute. Goldberg page 4

Back to Question 27

Answer 26

True

Explanation: Goldberg page 4

Back to Question 26

Answer 25

False

Explanation: An XML document can have only one root element. Goldberg page 4

Back to Question 25

Answer 24

True

Explanation: Goldberg page 4

Back to Question 24

Answer 23

True

Explanation: Goldberg page 4

Back to Question 23

Answer 22

True

Explanation: Goldberg page 4

Back to Question 22

Answer 21

False

Explanation: XML documents are saved with a .xml extension. Goldberg page 4

Back to Question 21

Answer 20

True

Explanation: Goldberg page 3

Back to Question 20

Answer 19

False

Explanation: XML is not a language itself. Rather, an XML document is written in a *custom markup language*, according to the XML specification. Goldberg page 3

Back to Question 19

Answer 18

True

Explanation: Goldberg page xv

Back to Question 18

Answer 17

True

Explanation: Goldberg page xv

Back to Question 17

Answer 16

False

Explanation: You can define the structure of an XML document by using a DTD *(Document Type Definition)* or with the XML Schema language. Goldberg page xiv

Back to Question 16

Answer 15

False

Explanation: XSL is made up of three languages: XSLT, XPath, and XSL-FO. Goldberg page xiv

Back to Question 15

Answer 14

True

Explanation: Goldberg page xiv

Back to Question 14

Answer 13

False

Explanation: Because there are no predefined tags, a browser cannot know how to display an XML document according to the author's wishes. It's

your job as the author of the document to specify how the document should be displayed. Goldberg page xiv

Back to Question 13

Answer 12

True

Explanation: Goldberg page xiv

Back to Question 12

Answer 11

False

Explanation: XML is a non-proprietary specification that is free to anyone who wishes to use it. Goldberg page xiii

Back to Question 11

Answer 10

False

Explanation: An XML document is a well-structured text file that is considered to be human-readable. Goldberg page xiii

Back to Question 10

Answer 9

True

Explanation: Goldberg page xiii

Back to Question 9

Answer 8

False

Explanation: XML is a language for storing and carrying information while HTML is a language for displaying information. Goldberg page xiii

Back to Question 8

Answer 7

False

Explanation: The XML specification makes it possible for people to define their own markup languages so that so that they or others can create XML documents using that markup language. Goldberg page xii

Back to Question 7

Answer 6

True

Explanation: Goldberg page xii

Back to Question 6

Answer 5

True

Explanation: Goldberg page xii

Back to Question 5

Answer 4

False

Explanation: XML has no predefined tags. The person writing the XML can create whatever tags they need provided that those tags adhere to the XML specification. Goldberg page xii.

Back to Question 4

Answer 3

True

Explanation: Goldberg page xii

Back to Question 3

Answer 2

True

Explanation: Goldberg page xii

Back to Question 2

Answer 1

False

Explanation: XML is an abbreviation for eXtensible Markup Language. Goldberg page xii.

Back to Question 1

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Xml0100 Writing XML Review
- File: Xml0100WritingXmlReview.htm
- Published: 11/08/13
- Revised: 08/21/15

on cnx.org and that it was made and published without my prior knowledge.
Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Xml0110 Transforming XML Review
This module contains review questions, answers, and explanations keyed to Chapter 2 of the textbook titled XML: Visual QuickStart Guide, 2nd Edition. By Kevin Howard Goldberg.

Table of Contents

Preface

This material is published in support of the course identified as ITSE 1356 - Extensible Markup Language (XML) at Austin Community College in Austin, TX. As of the Spring 2014 semester, the textbook for this course is XML: Visual QuickStart Guide, 2nd Edition. By Kevin Howard Goldberg.

This module contains review questions, answers, and explanations keyed to Chapter 2 of the textbook.

The questions and the answers are connected by hyperlinks to make it easy for you to navigate from the question to the answer and back again.

Questions

Question 1 .

True or False: Transforming an XML document means using XML to analyze its contents and then take certain actions depending on what

elements are found.

Answer 1

Question 2

True or False: The transformation process starts with two documents:

1. the XML document containing the source data to be transformed
2. the XSLT stylesheet document describing the rules of the transformation

Answer 2

Question 3

True or False: An XSLT transformation requires an XSLT processor.

Answer 3

Question 4

True or False: An XML document must always include an xml-stylesheet processing instruction in order to undergo an XSLT transformation.

Answer 4

Question 5

True or False: Listing 1 below shows the proper syntax for an xml-stylesheet processing instruction.

Listing 1 . Question 5.

```
<?xml-stylesheet type="text/xsl"
href="MyFile.xsl"?>
```

Answer 5

Question 6

True or False: A node tree is a hierarchical representation of the XSL document.

Answer 6

Question 7

True or False: A node in a node tree is one individual piece of the XML document such as an element, an attribute, or some text content.

Answer 7

Question 8

True or False: An XSLT style sheet contains instructions on what should be done with the nodes in a node tree in order to perform an XSLT transformation.

Answer 8

Question 9

True or False: Each XSLT template contains three parts:

1. a label that identifies the nodes to which the template applies
2. instructions about the transformation that should take place
3. identification of the XML document to which the transformation is to be applied

Answer 9

Question 10

True or False: XSLT templates cannot contain literal elements.

Answer 10

Question 11

True or False: XSLT style sheets are text files and are saved with an extension of .xml.

Answer 11

Question 12

True or False: XSLT uses the XPath language to identify nodes.

Answer 12

Question 13

True or False: An XSLT style sheet is actually an XML document.

Answer 13

Question 14

True or False: An XSLT style sheet does not require an XML declaration.

Answer 14

Question 15

True or False: In an XSLT style sheet, the XML declaration should be followed by the W3C namespace for style sheets.

Answer 15

Question 16

True or False: In an XSLT style sheet, the W3C namespace for style sheets must be followed by the root template.

Answer 16

Question 17

True or False: Listing 2 below shows the proper format for the opening tag of the root template.

Listing 2 . Question 17.

```
<xsl:template match="/">
```

Answer 17

Question 18

True or False: The XSLT style sheet must contain an ***xsl:output*** processing instruction with the format shown in Listing 3 below.

Listing 3 . Question 18.

```
<xsl:output method="html"/>
```

Answer 18

Question 19

True or False: All XSLT style sheets must be well-formed.

Answer 19

Question 20

True or False: An ***xsl:value-of*** element is used to access and output the contents of an XML document.

Answer 20

Question 21

True or False: You can use ***select="."*** in an ***xsl:value-of*** element to output the contents of the current node.

Answer 21

Question 22

True or False: The ***xsl:value-of*** element will only act on one node, even if it matches many nodes.

Answer 22

Question 23

True or False: The ***xsl:for-all*** element allows you to act on all matching nodes.

Answer 23

Question 24

True or False: You can use an ***xsl:if*** element to process a node or set of nodes only a certain condition is met.

Answer 24

Question 25

True or False: The general syntax for processing nodes conditionally is shown in Listing 4 below where *expression* specifies a node set, a string, or a number.

Listing 4 . Question 25.

```
<xsl:if test="expression">
<!-- insert processing code here -->
</xsl:if>
```

Answer 25

Question 26

True or False: You can use an ***xsl:choose*** element to test for several different conditions and react appropriately for each condition.

Answer 26

Question 27

True or False: An ***xsl:choose*** element must be nested inside an ***xsl:when*** element.

Answer 27

Question 28

True or False: You can use an ***xsl:sort*** element contained in an ***xsl:for-each*** element to cause nodes to be processed in a particular order.

Answer 28

Question 29

True or False: You cannot nest an ***xsl:sort*** element inside another ***xsl:sort*** element.

Answer 29

Question 30

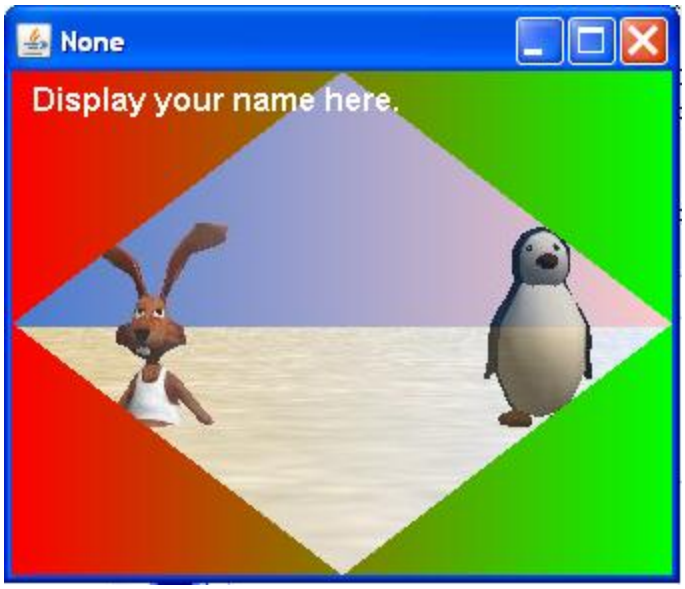
True or False: An XSLT style sheet can contain only one template, which is the root template.

Answer 30

Listings

What is the meaning of the following two images?

These images were inserted here simply to insert some space between the questions and the answers to keep them from being visible on the screen at the same time.



This image was also inserted for the purpose of inserting space between the questions and the answers.

Answers

Answer 30

False

Explanation: Goldberg page 34

Back to Question 30

Answer 29

False

Explanation: You can nest ***xsl:sort*** elements within other ***xsl:sort*** elements to sort on multiple keys. Goldberg page 32

Back to Question 29

Answer 28

True

Explanation: Goldberg page 32

Back to Question 28

Answer 27

False

Explanation: The order is reversed. An ***xsl:when*** element must be nested inside an ***xsl:choose*** element. Goldberg page 31

Back to Question 27

Answer 26

True

Explanation: Goldberg page 31

Back to Question 26

Answer 25

True

Explanation: Goldberg page 30

Back to Question 25

Answer 24

True

Explanation: Goldberg page 30

Back to Question 24

Answer 23

False

Explanation: The ***xsl:for-each*** element allows you to act on all matching nodes. Goldberg page 28

Back to Question 23

Answer 22

True

Explanation: Goldberg page 28

Back to Question 22

Answer 21

True

Explanation: Goldberg page 27

Back to Question 21

Answer 20

True

Explanation: Goldberg page 26

Back to Question 20

Answer 19

True

Explanation: Goldberg page 25

Back to Question 19

Answer 18

False

Explanation: The ***xsl:output*** processing instruction can set the output method to "html", "xml", or "text". The ***xsl:output*** processing instruction can also be omitted, in which case the XSLT processor will output XML by default. Goldberg page 24

Back to Question 18

Answer 17

True

Explanation: Goldberg page 23

Back to Question 17

Answer 16

False

Explanation: The XSLT processor doesn't care where the root template appears in your XSLT style sheet. Goldberg page 23

Back to Question 16

Answer 15

True

Explanation: Goldberg page 22

Back to Question 15

Answer 14

False

Explanation: An XSLT style sheet is actually an XML document and should begin with a standard XML declaration. Goldberg page 22

Back to Question 14

Answer 13

True

Explanation: Goldberg page 22

Back to Question 13

Answer 12

True

Explanation: Goldberg page 21

Back to Question 12

Answer 11

False

Explanation: XSLT style sheets are text files and are saved with an .xsl extension. Goldberg page 21

Back to Question 11

Answer 10

False

Explanation: XSLT templates can contain literal elements that should be output exactly as written. Goldberg page 21

Back to Question 10

Answer 9

False

Explanation: The third item in the list is not part of an XSLT template. Goldberg page 20

Back to Question 9

Answer 8

True

Explanation: Goldberg page 20

Back to Question 8

Answer 7

True

Explanation: Goldberg page 20

Back to Question 7

Answer 6

False

Explanation: A node tree is a hierarchical representation of the **XML** document. Goldberg page 20

Back to Question 6

Answer 5

True

Explanation: Goldberg page 20

Back to Question 5

Answer 4

False

Explanation: While an xml-stylesheet processing instruction is often required, with some XSLT processors, you don't need an xml-stylesheet processing instruction in your XML document. Goldberg page 21

Back to Question 4

Answer 3

True

Explanation: Goldberg page 20

Back to Question 3

Answer 2

True

Explanation: Goldberg page 20

Back to Question 2

Answer 1

False

Explanation: Transforming an XML document means using **XSLT** to analyze its contents and then take certain actions depending on what elements are found. Goldberg page 19

Back to Question 1

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Xml0110 Transforming XML Review

- File: Xmi0110TransformingXML.htm
- Published: 11/08/13
- Revised: 12/02/14

Note: Disclaimers:
Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.
I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.
Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Xml0120 Validating XML Review
This module contains review questions, answers, and explanations keyed to Chapters 6, 7, and 8 of the textbook titled XML: Visual QuickStart Guide, 2nd Edition. By Kevin Howard Goldberg.

Table of Contents

Preface

This material is published in support of the course identified as ITSE 1356 - Extensible Markup Language (XML) at Austin Community College in Austin, TX. As of the Spring 2014 semester, the textbook for this course is XML: Visual QuickStart Guide, 2nd Edition. By Kevin Howard Goldberg.

This module contains review questions, answers, and explanations keyed to the following chapters of the textbook:

- Chapter 6 except for the following sections:
    - Referencing Attributes with Unique Values
    - Restricting Attributes to Valid XML Names
- Chapter 7 except for the following sections:
    - Creating Entities for Unparsed Content
    - Embedding Unparsed Content
- Chapter 8 except for the following sections:

    - Naming a Public External DTD
    - Declaring a Public External DTD
    - Pros and Cons of DTDs

The questions and the answers are connected by hyperlinks to make it easy for you to navigate from the question to the answer and back again.

Questions

Question 1 .

True or False: A ***schema*** for a custom markup language:

- identifies its elements and their attributes
- declares which are required and which are not

Answer 1

Question 2

True or False: Every XML document requires a schema.

Answer 2

Question 3

True or False: You can compare an XML document to its corresponding schema to *validate* whether it conforms to the rules specified in the schema.

Answer 3

Question 4

True or False: There are two principal systems for writing schemas: DTD and PDQ.

Answer 4

Question 5

True or False: DTD is an abbreviation for *Document to Document* .

Answer 5

Question 6

True or False: A DTD is a set of rules that defines a custom markup language in XML.

Answer 6

Question 7

True or False: An XML document is considered *valid* for a particular custom markup language if it adheres to the rules defined by the DTD for that custom markup language.

Answer 7

Question 8

True or False: A DTD for a custom markup language will define:

1. a list of elements
2. any child elements that each element can have

3. any attribute that each element can have
4. whether or not the specified attributes are optional or required
5. whether the document is or is not sensitive to upper and lower case

Answer 8

Question 9

True or False: A DTD is an XML document that begins with a standard XML declaration.

Answer 9

Question 10

True or False: PCDATA is an abbreviation for *politically correct data* .

Answer 10

Question 11

True or False: You can use a DTD to control the order in which elements must appear in a corresponding XML document.

Answer 11

Question 12

True or False: A DTD cannot define a sequence of child elements that must be contained in a parent element.

Answer 12

Question 13

True or False: The three characters *, +, and ? are special symbols used in a DTD to define how many times a child element can appear within a parent element.

Answer 13

Question 14

True or False: The vertical bar character, |, is used in a DTD to define choices for the content of an element.

Answer 14

Question 15

True or False: In a DTD, an attribute definition consists of four parts:

1. element name
2. attribute name
3. attribute type
4. optional status

Answer 15

Question 16

True or False: *Internal general entities* defined in a DTD are shortcuts that represent text.

Answer 16

Question 17

True or False: To use an internal general entity in an XML document, you write its name with an ampersand as a prefix and a colon as a suffix.

Answer 17

Question 18

True or False: Character references look similar to entities but they are not entities and do not need to be declared in the DTD.

Answer 18

Question 19

True or False: An alternative to the *internal general entity* is an *external general entity* , which is saved in a separate, external document.

Answer 19

Question 20

True or False: Parameter entities are created for use in XSL documents.

Answer 20

Question 21

True or False: As with general entities, parameter entities can also be created in external files.

Answer 21

Question 22

True or False: You must declare a DTD in your XML document in order to use it.

Answer 22

Question 23

True or False: The main purpose for creating a DTD is to ensure that a given XML document is constructed in a specific way as defined by the DTD.

Answer 23

Question 24

True or False: All DTDs must be written and saved as separate files.

Answer 24

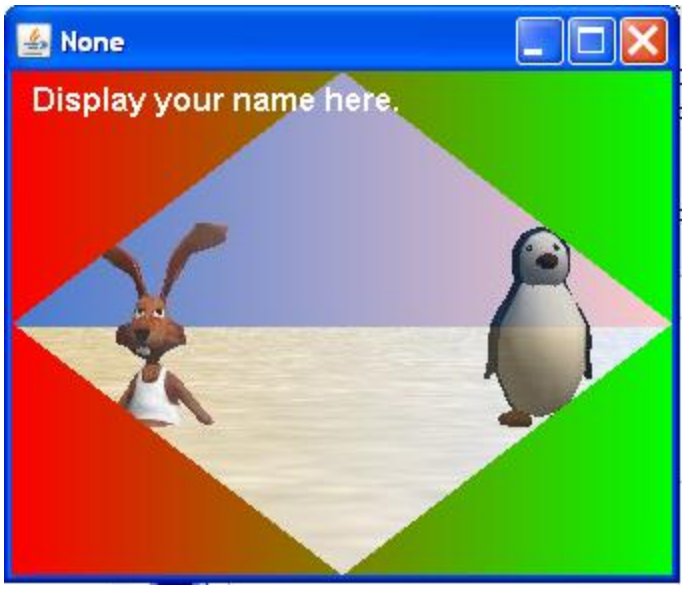
Question 25

True or False: All XML parsers are required to have the ability to validate your XML against a DTD.

Answer 25

What is the meaning of the following two images?

These images were inserted here simply to insert some space between the questions and the answers to keep them from being visible on the screen at the same time.



This image was also inserted for the purpose of inserting space between the questions and the answers.

Answers

Answer 25

False

Explanation: Goldberg page 107

Back to Question 25

Answer 24

False

Explanation: DTDs can be written and saved as separate files, or they can be written entirely inside an XML document. Goldberg page 103

Back to Question 24

Answer 23

True

Explanation: Goldberg page 103

Back to Question 23

Answer 22

True

Explanation: Goldberg page 103

Back to Question 22

Answer 21

True

Explanation: Goldberg page 101

Back to Question 21

Answer 20

False

Explanation: Parameter entities are created for the DTD itself. Goldberg page 100

Back to Question 20

Answer 19

True

Explanation: Goldberg page 94

Back to Question 19

Answer 18

True

Explanation: Goldberg page 93

Back to Question 18

Answer 17

False

Explanation: To use an internal general entity in an XML document, you write its name with an ampersand as a prefix and a **semicolon** as a suffix.

Back to Question 17

Answer 16

True

Explanation: Goldberg page 92

Back to Question 16

Answer 15

True

Explanation: Goldberg page 85

Back to Question 15

Answer 14

True

Explanation: Goldberg page 82

Back to Question 14

Answer 13

True

Explanation: Goldberg page 81

Back to Question 13

Answer 12

False

Explanation: A DTD **can** define a sequence of child elements that must be contained in a parent element. Goldberg page 80

Back to Question 12

Answer 11

True

Explanation: Goldberg page 79

Back to Question 11

Answer 10

False

Explanation: Goldberg page 77

Back to Question 10

Answer 9

False

Explanation: Goldberg page 76

Back to Question 9

Answer 8

False

Explanation: A DTD for a custom markup language will only define:

1. a list of elements
2. any child elements that each element can have
3. any attribute that each element can have
4. whether or not the specified attributes are optional or required

The fifth item in the list given in the question is not part of a DTD.

Back to Question 8

Answer 7

True

Explanation: Goldberg page 76

Back to Question 7

Answer 6

True

Explanation: Goldberg page 76

Back to Question 6

Answer 5

False

Explanation: DTD is an abbreviation for *Document Type Definition* .

Back to Question 5

Answer 4

False

Explanation: There are two principal systems for writing schemas: DTD and XML Schema.

Back to Question 4

Answer 3

True

Explanation: Goldberg page 75

Back to Question 3

Answer 2

False

Explanation: Goldberg page 75

Back to Question 2

Answer 1

True

Explanation: Goldberg page 75

Back to Question 1

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Xml0120 Validating XML Review
- File: Xml0120ValidatingXML.htm
- Published: 11/10/13
- Revised: 12/02/14

In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Json0110: Preface to JSON
This module is a preface to several modules on JSON.

Table of Contents

Welcome

Welcome to this set of introductory modules on JSON *(JavaScript Object Notation)* in my collection titled *Introduction to XML* .

ITSE1356 and JSON

ITSE1356

This is a portion of the material that I use to teach the course identified as **ITSE 1356 - Extensible Markup Language (XML)** at Austin Community College in Austin, TX.

As of the Spring 2014 semester, the textbook for this course is *XML: Visual QuickStart Guide, 2nd Edition* , By Kevin Howard Goldberg.

JSON

JSON is a lightweight data-interchange format. It is emerging as a strong alternative to the use of XML for data-interchange purposes.

An introductory section on JSON is scheduled to be introduced into the ITSE1356 course beginning in the Fall 2014 semester. JSON material is not included in the Goldberg textbook mentioned above. Therefore, these JSON modules will constitute the primary learning resource for the JSON section of the course.

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Json0110: Preface to JSON
- File: Json0110.htm
- Published: 02/02/14

Note: Disclaimers:
Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.
I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module.
In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.
Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Json0120: What is JSON?
JSON is a general purpose data interchange format that is supported by Java, PHP, JavaScript, and other programming languages. JSON is a standard that describes how ordered lists and unordered maps, strings boolean values and numbers can be represented as text in a string.

Table of Contents

Preface

This module is one in a collection of modules designed for teaching **ITSE 1356 Introduction to XML** at Austin Community College in Austin, TX.

The title of this module is *"What is JSON?"* Given that there are dozens of tutorials and blogs on the web that talk about JSON, you are probably wondering why we need another tutorial on JSON.

Specialized modules

The modules in this collection are very specialized. As indicated above, they are designed for teaching a very specific XML course at the college where I teach. That course is predominately based on XML and uses an XML textbook for the major portion of the course.

An alternative to XML

The online document titled Introducing JSON begins as follows:

> "JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is

> easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language."

Because JSON is an emerging alternative to XML, beginning in the Fall of 2014, the course will include an introductory section on JSON. The modules in this collection and the online resources pointed to by these modules will be the primary learning resource for the JSON portion of the course.

> JSON could conceivably be used for a variety of purposes. However, for the purposes of this course, it will be viewed solely as a data-interchange language and an alternative to the use for XML for that same purpose.

Background information

JSON is an acronym for **JavaScript Object Notation** . Don't be fooled by the name however. Although JSON is based on JavaScript object syntax, it is not intended to be JavaScript *(at least not exclusively JavaScript)* .

A general data interchange format

Instead, similar to XML, JSON is a general purpose data interchange format that is supported by Java, PHP, JavaScript, and other programming languages. JSON is a standard that describes how ordered lists and unordered maps, strings boolean values and numbers can be represented as text in a string.

Just like XML, JSON provides a way to pass structured information between languages or between different computing environments using the same language.

Typical operation

Typically a data construct, *(such as an object for example)* , in one programming environment will be transformed into a JSON string. That string will be transported to another programming environment where it will be transformed into a data construct, *(such as a hash table for example)* , that is suitable for use in that programming environment

A real-world analogy

Consider the following analogous situation. A young family has a large playscape for their children in their back yard. They need to move to another house across town. In order to save money, they rent a small truck and do the entire move themselves.

A playscape object

The playscape can be thought of as an object with certain properties such as **swing** and **slide** .

It is too large to fit into the truck so the adults disassemble it into a well-organized package of boards, chains, bolts, nuts, etc. They are very careful to label each part and to create some drawings showing the organization of the parts for use later.

No longer an object

In that disassembled state, it can no longer be thought of as an object with properties of **swing** and **slide** . Instead, it is simply a well-organized and documented package of parts. The package of parts is analogous to a JSON string. The playscape object has been transformed into a well-organized package of parts.

Reassemble the parts

After the parts are transported to the new location, they are reassembled into an object with properties of **swing** and **slide** .

This is what we do with JSON. We disassemble an object *(or other data construct)* into a JSON string: a well-organized package of parts. Later on, and possibly in an entirely different programming environment, we reassemble the parts into a data construct suitable for use in the new programming environment.

Streamlined procedures

JavaScript, Java, PHP, and other programming languages provide streamlined procedures for transforming a data construct into a JSON string and for transforming a JSON string into a suitable data construct. As an example, the **JSON.stringify** method can be used to transform a JavaScript object into a JSON string. The **JSON.parse** method can be used to transform a JSON string into a JavaScript object. Other methods or functions are available to accomplish the same purposes in other languages.

Online references

There are many good online JSON references. Here are a few:

- Introducing JSON
- JSON: The Fat-Free Alternative to XML
- JSON with PHP
- JSON in Java
- Java API for JSON Processing: An Introduction to JSON
- JSON tutorial for beginners learn how to program part 1 JavaScript (video)
- JSON in JavaScript
- JSON: What It Is, How It Works, How to Use It

What's next?

The next module will present and explain some sample scripts that show how to use JSON with JavaScript.

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Json0120: What is JSON?
- File: Json0120.htm
- Published: 02/02/14

Note: Disclaimers:
Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.
I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module.
In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.
Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Json0130: JSON and JavaScript
Learn how to transform a JavaScript object into a JSON string and how to transform the JSON string back into a JavaScript object.

Table of Contents

Preface

This module is one in a collection of modules designed for teaching **ITSE 1356 Introduction to XML** at Austin Community College in Austin, TX.

As mentioned in an earlier module, because JSON is an emerging alternative to XML, beginning in the Fall of 2014, the course will include an introductory section on JSON. The modules in this collection and the online resources pointed to by these modules will be the primary learning resource for the JSON portion of the course.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the Figures and Listings while you are reading about them.

Figures

Listings

Preview

I will present and explain three JavaScript scripts in this module. Each script shows how to transform a JavaScript object into a JSON string and how to transform the JSON string back into a JavaScript object.

The first script deals with a very simple JavaScript object. The second script deals with a slightly more complex JavaScript object that encapsulates an array of JavaScript objects.

The third script illustrates how to tell the difference between a JavaScript object in *object literal* format and a JSON string.

In addition to the three scripts described above, I will provide a word of caution regarding a possible terminology issue on a major technical website using a script from that website as an example of the issue.

General background information

Prerequisite knowledge

In order to understand the material in this module, you will need a moderate understanding of HTML and JavaScript programming.

As you learned in an earlier module, several different programming languages including JavaScript, Java, and PHP support JSON. I elected to use JavaScript in these modules because many of the students enrolled in this course are also enrolled in the web development curriculum at Austin Community College. Those students either already have, or shortly will have a requirement to learn HTML and JavaScript programming for their other coursework.

If you already know HTML and you know how to program using JavaScript, you should continue with this module. If not, you need to take a side trip and learn how to program in JavaScript before continuing. Also, if you don't know HTML, you should learn that also.

There are hundreds of online tutorials available for learning HTML and JavaScript, some better than others. For HTML, I recommend the free online *Introduction to HTML* tutorial at http://www.codecademy.com/tracks/web . For JavaScript, I recommend the free *JavaScript* tutorial at http://www.codecademy.com/tracks/javascript An average college student should be able to complete either tutorial in about fifteen hours or less. Once you complete either or both tutorials, your knowledge of HTML and JavaScript should be sufficient for an understanding of JSON.

Differences between a JSON string and a JavaScript object

The syntax of a JSON string looks a lot like the syntax of a JavaScript object in ***object literal notation*** .

> Compare the JSON string in the second line in Figure 2 with the object literal in the second line in Listing 2 .

This can be confusing to those who don't recognize the difference between the two. I will explain some of the differences in this module.

A JavaScript object is a type

A JavaScript object encapsulates data and methods and exhibits behavior. Despite the similarity of appearance, a JSON string is simply a string of characters with a well-defined format. It is not a type and it does not exhibit behavior.

> At least it doesn't exhibit any behavior that is not expected of any other string.

Remember the playscape?

Harkening back to an earlier module, a JavaScript object is analogous to the playscape in the back yard that has **swing** and **slide** properties. The playscape can "do something" that many children find enjoyable.

A JSON string is analogous to the well-organized package of parts resulting from the disassembly of the playscape. In its disassembled state, the JSON string does not have properties like **swing** and **slide** even though all of the parts necessary to support those properties are in the package of parts. In that state, all it can do is lay there and take up space. It is unlikely that children would find it to be enjoyable.

I will refer back to the playscape a few more times in this module.

Transform a JSON string into a JavaScript object

In order to do much in the way of significant processing on the contents of a JSON string using JavaScript, you first need to transform it into a JavaScript object. *(You need to reassemble the playscape before the children can play on it.)*

> If you are working in some other language, you need to transform it into a data structure that is appropriate for that language.

There are different ways to transform a JSON string into a JavaScript object, at least one of which has security problems. *(The **eval** function is said to have security problems.)*

The JSON.parse method

The recommended way to transform a JSON string into a JavaScript object is to call the **JSON.parse** method passing the JSON string as a parameter.

> The JSON.parse method is apparently supported by most if not all modern browsers.

The **JSON.parse** method returns the JavaScript object that corresponds to the JSON string. I will use this method in the JavaScript scripts that I will explain later.

The JSON.stringify method

On the flip side of the coin, if you need to disassemble that JavaScript playscape object into a well-organized package of parts, you can do that by calling the **JSON.stringify** method passing the JavaScript object as a parameter. The **JSON.stringify** method returns the JSON string that represents the object, and amazingly does that even for very complex JavaScript objects.

> However, the simple form of the JSON.stringify method that I will use in this module does not preserve methods that may reside in the JavaScript object.

With that as an introduction, let's look at some code.

Discussion and sample code

Json0130a.htm

I will explain the three scripts using fragments of code. The first script that I will explain is shown in its entirety in Listing 16 . If you open the file named **Json0130a.htm** in Firefox v26 or a later version, the text shown in Figure 1 should be displayed in the browser window.

Figure 1 . Screen output from Json0130a.htm.

Figure 1 . Screen output from Json0130a.htm.

```
Create a JavaScript object.
Unsuccessful attempt to display object.
[object Object]
Display keys in object: name age method
Display values in object: Bill, 31, true

Transform JavaScript object into a JSON string.
Display JSON string {"name":"Bill","age":31}
Unsuccessful attempt to access name and age.
undefined, undefined

Transform the JSON string into a JavaScript object.
Display values in object: Bill, 31
Display keys in object: name age
```

(Note that I manually inserted some blank lines in Figure 1 to make it easier on the eyes.)

I will refer back to the contents of Figure 1 as I explain the code fragments.

Define a JavaScript function

Listing 1 shows the beginning of the script and also shows the definition of a simple JavaScript function that will be included as part of a JavaScript object. You will see shortly, however, that even though this function is part of the object, it is discarded when the object is transformed into a JSON string using the simple version of the **JSON.stringify** method. *(More complicated versions of the **JSON.stringify** method are available in some browsers.)*

Listing 1 . Define a JavaScript function.

Listing 1 . Define a JavaScript function.

```
<body>
  <script>
    //Define a function
    function aMethod(){return true;};
```

There is nothing new or exciting about the code in Listing 1 . This is "plain vanilla" JavaScript programming.

Create a JavaScript object

The code in Listing 2 creates a new JavaScript object containing two properties (*name and age)* and a method using *object literal notation* .

Listing 2 . Create a JavaScript object.

```
    document.write("Create a JavaScript object.");
    var obj01 = {name:"Bill",age:31,method:aMethod};
```

Recall that there is at least one other way to create an object in JavaScript by using a constructor. I elected this approach because this is the syntax that is most easily confused with the syntax of a JSON string.

Explanatory output text

The first line of code in Listing 2 produces the first line of output text in Figure 1 . Similar statements will be included at critical points in the script so that the output will be somewhat self explanatory.

Garbage out

The code in Listing 3 was included to show that unlike in some other programming languages such as Java, simply passing a JavaScript object to the **document.write** method does not produce a meaningful output.

Listing 3 . Garbage out.

```
      document.write(
           "<br/>Unsuccessful attempt to display
object.");
      document.write("<br/>" + obj01);
```

The code in Listing 3 produced the second and third lines of output text in Figure 1 .

Display keys in object

The code in Listing 4 displays the three keys in the object, producing the fourth line of text in Figure 1 .

Listing 4 . Display keys in object.

```
      document.write("<br/>Display keys in object: ");
      for (var key in obj01) {
        if (obj01.hasOwnProperty(key)) {
          document.write(key + " ");
        }//end if
      }//end for loop
```

You may *(or may not)* need to refresh your memory of JavaScript programming to understand the code in Listing 4 . In any event, so far all the code that we have seen is "plain vanilla" JavaScript code.

Display object's values

Let's see one more section of "plain vanilla" code before we get into the good stuff. Listing 5 uses dot notation to display the values of the properties in the object producing the output on line 5 in Figure 1 .

Listing 5 . Display object's values.

```
      document.write("<br/>Display values in object: ");
      document.write(obj01.name + ", "
                     + obj01.age + ", " +
obj01.method());
```

Note that the value displayed for the key named **method** is the result of evaluating the method: **true** .

Cleanup time

It's getting a little difficult to find the referenced lines in Figure 1 . To make the output text a little easier to find, Figure 2 contains the output text lines from Figure 1 that haven't been discussed yet.

Figure 2 . Partial screen output from Json0130a.htm.

```
Transform JavaScript object into a JSON string.
Display JSON string {"name":"Bill","age":31}
Unsuccessful attempt to access name and age.
undefined, undefined

Transform the JSON string into a JavaScript object.
Display values in object: Bill, 31
Display keys in object: name age
```

(Note that once again I manually inserted a blank line in Figure 2 to make it easier on the eyes.)

The good stuff

Finally, we are ready to see something new and interesting. The call to the **JSON.stringify** method in Listing 6 transforms the JavaScript object that was created in Listing 2 into a JSON string and produces the first two lines of output text in Figure 2 .

Listing 6 . Transform JavaScript object into a JSON string.

```
      document.write("<br/>Transform JavaScript object "
+
                                    "into a JSON
string.");
      // Note that the method does not become part of the
      // JSON string.
      var jsonstring = JSON.stringify(obj01);
      document.write(
                "<br/>Display JSON string " + jsonstring
);
```

The new JSON string is saved in the variable named **jsonstring** .

The method has been lost

First note that as indicated earlier, this simple version of the **JSON.stringify** method discards methods belonging to the object when transforming it to a JSON string. Therefore, from this point forward in the script, the method belonging to the original JavaScript object has been lost.

Note the similarity

Once again, note the similarity between the JSON string shown in the second line of Figure 2 and the object literal version of the JavaScript object shown in Listing 2 .

Having discarded the method, the only difference between the two is that the keys in the JSON string are enclosed in quotes while the keys in Listing 2 are not enclosed in quotes.

> As I understand it, the keys in a JSON string must always be enclosed in quotes while quotes are normally optional for keys in the *object literal* declaration of a JavaScript object. *(Some keys must be enclosed in quotes in the object literal syntax for a JavaScript object.)*

Unsuccessful attempt to access name and age

However, even though the syntax is very similar, a JSON string is very different from a JavaScript object. A JavaScript object is a type having content and behavior. A JSON string

is just a string of characters having content but no behavior. This is illustrated by the code in Listing 7 , which is very similar to the code in Listing 5 .

Listing 7 . Unsuccessful attempt to access name and age.

```
      document.write("<br/>Unsuccessful attempt to " +
                                    "access name and
age.");
      document.write("<br/>" + jsonstring.name + ", "
                                              +
jsonstring.age)
```

When dot notation was used to access the **name** and **age** properties of the object in Listing 5 , the values of those properties were returned and displayed on Line 5 in Figure 1 .

When a similar syntax was used in an attempt to access the values associated with name and age in Listing 7 , the result was "undefined" as shown on the fourth line of Figure 2 . In other words, a JSON string is just what it says; simply a string of characters.

The magic of a JSON string

To the extent that there may be magic, the magic of the JSON string is

- the way that the characters are organized to represent the object that was used to create the string, and
- the ability to use the characters in that string to replicate the object at a later time and possibly in a different location and different programming environment.

Cleanup time again

Let's create one more simplified Figure showing the script output. Figure 3 shows the last three lines of text from Figure 1 that haven't been discussed yet.

Figure 3 . Partial screen output from Json0130a.htm.

```
Transform the JSON string into a JavaScript object.
Display values in object: Bill, 31
Display keys in object: name age
```

Transform the JSON string into a JavaScript object

The first two lines of text in Figure 3 were produced by the code in Listing 8 .

Listing 8 . Transform the JSON string into a JavaScript object.

```
      document.write("<br/>Transform the JSON string " +
                              "into a JavaScript
object.");
      var obj02 = JSON.parse(jsonstring);
      document.write("<br/>Display values in object: ");
      document.write(obj02.name + ", " + obj02.age);
```

Let's pretend

For simplicity, I combined the disassembly and the reassembly of the object into a single script. Let's pretend, however, that the code discussed down to this point resides on Computer-A and the remaining code resides on Computer-B at a different location. Pretend that the JSON string created in Listing 6 has been transported from Computer-A to Computer-B. Now its time for the code in Computer-B to use that JSON string to replicate the original object.

Transform the JSON string into a JavaScript object

The code in Listing 8 calls the **JSON.parse** method passing the JSON string as a parameter to create a replica of the original JavaScript object that was created on Computer-A.

The **JSON.parse** method uses the JSON string to reassemble and return an object that is a replica of the original object that was created in Listing 2 *(minus the method property, which was lost in the creation of the JSON string in Listing 6)* .

Then Listing 8 uses code similar to that shown in Listing 5 to access and display the values of the remaining two properties of the object as shown by the second line in Figure 3 .

Finally, the code in Listing 9 uses code similar to that shown earlier in Listing 4 to display the keys in the new object. The result is shown in the last line in Figure 3 where the keys are **name** and **age** with the **method** key missing.

Listing 9 . Display keys in object.

```
      document.write("<br/>Display keys in object: ");;
      // Note that it does not contain the method from
the
      // original JavaScript object.
      for (var key in obj02) {
        if (obj02.hasOwnProperty(key)) {
          document.write(key + " ");
        }//end if
      }//end for loop

    </script>
  </body>
```

Listing 9 also signals the end of the script.

Json0130b.htm

Listing 17 presents a similar but slightly more complicated script than the one discussed earlier. As before, I will discuss the code in fragments.

Output from the script

Figure 4 shows the screen output produced by opening this file in Firefox v26 or later.

Once again, I manually inserted some blank lines to make it easier on the eyes.

Figure 4 . Screen output from Json0130b,htm.

```
Create a JavaScript object involving array data.
Display values in object.
Bill:31
Jill:40

Transform the object into a JSON string.
Display JSON string.
{"friends":[{"name":"Bill","age":31},
{"name":"Jill","age":40}]}

Transform the JSON string into a JavaScript object.
Display values in object.
Bill:31
Jill:40
```

Create a JavaScript object

Listing 10 shows the creation of a JavaScript object and the display of the property values contained in that object.

Listing 10 . Create a JavaScript object.

Listing 10 . Create a JavaScript object.

```
  <body>
    <script>
      document.write("Create a JavaScript object " +
                                    "involving array
data.");
      var obj01 = {friends:[
              {name:"Bill",age:31},
{name:"Jill",age:40}]};
      document.write("<br/>Display values in object.");
      document.write("<br/>" + obj01.friends[0].name +
                                    ":" +
obj01.friends[0].age);
      document.write("<br/>" + obj01.friends[1].name +
                                    ":" +
obj01.friends[1].age);
```

One property is an array

This JavaScript object contains a property with a key named **friends** . The value of the friends property is a JavaScript array containing two JavaScript objects. Therefore, Listing 10 uses square bracket ([]) notation to access and display the values in the objects that are elements of the array.

The code in Listing 10 produces the first four lines of output text in Figure 4 .

Transform the object into a JSON string

Listing 11 transforms the JavaScript object into a JSON string and displays the string.

Listing 11 . Transform the object into a JSON string.

Listing 11 . Transform the object into a JSON string.

```
      document.write(
          "<br/>Transform the object into a JSON
string.");
      var jsontext = JSON.stringify(obj01);
      document.write("<br/>Display JSON string.");
      document.write("<br/>" + jsontext);
```

This produces the three lines of output text in the group near the center of Figure 4 . Note the similarity of the string shown in Figure 4 and the object literal code used to create the JavaScript object in Listing 10 .

> At the risk of becoming boring, I will state once again that even though they may look alike, a JSON string is different from a JavaScript object. A JavaScript object typically has properties and behavior. A JSON string is simply a package of characters and has no properties or behavior other than those that may typically be ascribed to any string of characters.

Transform the JSON string into a JavaScript object

Finally, Listing 12 uses the **JSON.parse** method to transform the JSON string into a JavaScript object and displays the values of the new object's properties. The object named **obj02** is a replica of the original object named **obj01** that was created in Listing 10 .

Listing 12 . Transform the JSON string into a JavaScript object.

Listing 12 . Transform the JSON string into a JavaScript object.

```
      document.write("<br/>Transform the JSON string " +
                              "into a JavaScript
object.");
      var obj02 = JSON.parse(jsontext);
      document.write("<br/>Display values in object.");
      document.write("<br/>" + obj02.friends[0].name
                              + ":" +
obj02.friends[0].age);
      document.write("<br/>" + obj02.friends[1].name
                              + ":" +
obj02.friends[1].age);

    </script>
  </body>
```

The code in Listing 12 produces the bottom four lines of output text shown in Figure 4 .

Listing 12 also signals the end of the script.

A word of caution

At this point, I want to alert you to a possible terminology issue that you may encounter while searching the web for information about JSON.

Possible terminology issue

Listing 13 contains a scaled down version of code that I copied from http://www.w3schools.com/json/json_intro.asp plus some code that I added myself.

Listing 13 . Possible terminology issue.

Listing 13 . Possible terminology issue.

```
<!DOCTYPE html>
<html>
<body>
<h2>JSON Object Creation in JavaScript</h2>

<p>
Name: <span id="jname"></span><br />
Age: <span id="jage"></span><br />
</p>

<script>
var JSONObject= {
"name":"John Johnson",
"age":33};
document.getElementById("jname").innerHTML=JSONObject.name
document.getElementById("jage").innerHTML=JSONObject.age

document.write("Code added by Baldwin");
var JSONstring = JSON.stringify(JSONObject);
document.write("<br/> " + JSONstring);
var JSObject = JSON.parse(JSONstring);
document.write("<br/> " + JSObject.name + ", " +
JSObject.age);
</script>

</body>
</html>
```

The variable named JSONObject

Despite the name, the variable named **JSONObject** in Listing 13 appears to be an ordinary JavaScript object in *object literal* format that has nothing to do with JSON.

> The keys "**name**" and "**age**" in Listing 13 are enclosed in double quotes, which may be a little unusual, but is perfectly valid for a JavaScript object. Although JSON keys must be enclosed in quotes, enclosing JavaScript object keys in quotes does not produce a JSON string.

A terminology issue?

This is probably just a terminology issue. However, I don't see anything in the original script at w3schools.com that illustrates anything about JSON.

Stringify the JavaScript object

To illustrate that **JSONObject** is a JavaScript object *(and is not JSON text)* , the code in Listing 13 converts it into a JSON string by passing it to the **JSON.stringify** method.

After discussing the parsing of JSON text, the document at JSON in JavaScript states

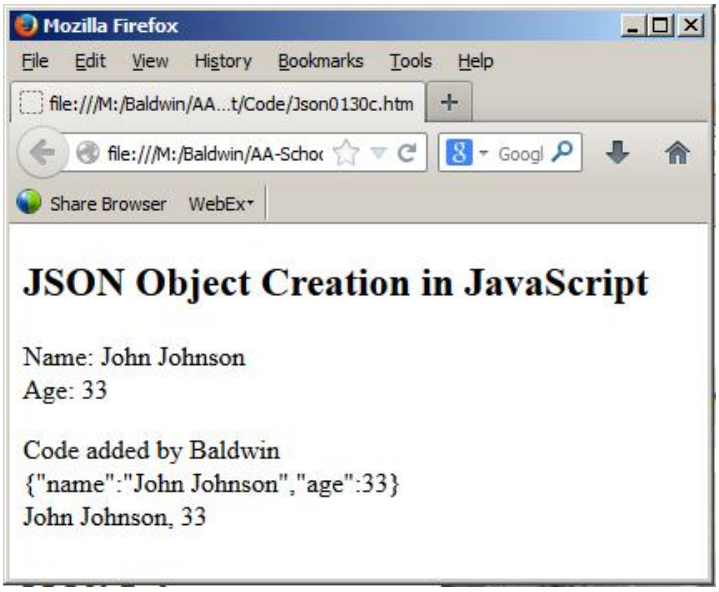
> "A JSON stringifier goes in the opposite direction, converting JavaScript data structures into JSON text."

The fact that **JSON.stringify** will accept **JSONObject** as an incoming parameter and return a JSON string seems to confirm that **JSONObject** is a JavaScript data structure *(an object)* .

Parse the JSON string

After displaying the JSON string, Listing 13 calls the **JSON.parse** method, passing the JSON string as a parameter, to replicate the original JavaScript object and displays the values of the object's properties. The screen output is shown in Figure 5 .

Figure 5 . Possible terminology issue.



Be wary of "JSON objects"

So, the word of caution is, be wary of material that refers to JSON objects. According to Introducing JSON

> "JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language."

The reason that JSON is a text format is probably the same reason that XML is a text format. The use of a text format *(as opposed to some proprietary object format)* ensures that the format can be read by almost any programming language running on almost any computer.

Recognizing the difference

Given all of the above, you might be wondering how to distinguish between a JavaScript object in *object literal* format and a JSON string. It all comes down to syntax.

Listing 18 presents a simple script that illustrates the difference. Once again, I will discuss it in fragments.

A comparison

Listing 14 shows a JavaScript object in *object literal* format and a JSON string on two consecutive lines to make them easy to compare.

Listing 14 . Comparison of object and string.

Listing 14 . Comparison of object and string.

```
<body>
  <script>
    var jScrObj01 =  {"name":"John","age":33};
    var jSonStr01 = '{"name":"Bill","age":33}';

    document.write("<br/>1. " + jScrObj01.name);
    document.write("<br/>2. " + jSonStr01.name);

    document.write("<br/>3. " + jScrObj01);
    document.write("<br/>4. " + jSonStr01);
```

The variable named **jScrObj01** is *(or contains)* a JavaScript object.

The variable named **jSonStr01** is *(or contains)* a JSON string.

The only difference between the two is the pair of single quotes that surrounds the expression on the right side of the assignment operator for **jSonStr01** .

> The keys in the JavaScript object are surrounded by double quotes. This is optional. If those quotes were removed, the JavaScript object would still be a JavaScript object.

Output from the script

After creating the JavaScript object and the JSON string, the code in Listing 14 executes four **write** statements to display information about the object and the string. The results are shown in Figure 6 .

Figure 6 . Output from the script.

Figure 6 . Output from the script.

```
1. John
2. undefined
3. [object Object]
4. {"name":"Bill","age":33}
```

Without getting into the details, the output in Figure 6 confirms that **jScrObj01** contains a JavaScript object and **jSonStr01** contains a JSON string.

Transform them both

The code in Listing 15

- calls the **JSON.parse** method to transform the JSON string into a JavaScript object, and
- calls the **JSON.stringify** method to transform the JavaScript object into a JSON string.

Listing 15 . Transform them both.

```
    var jScrObj02 = JSON.parse(jSonStr01);
    var jSonStr02 = JSON.stringify(jScrObj01);

    document.write("<br/>5. " + jScrObj02.name);
    document.write("<br/>6. " + jSonStr02.name);

    document.write("<br/>7. " + jScrObj02);
    document.write("<br/>8. " + jSonStr02);
  </script>

</body>
```

Output from the script

After that, the code in Listing 15 executes four **write** statements to display information about the new object and the new string. The results are shown in Figure 7 .

Figure 7 . Output from the script.

```
5. Bill
6. undefined
7. [object Object]
8. {"name":"John","age":33}
```

Once again, without getting into the details, the output in Figure 7 confirms that **jScrObj02** is a JavaScript object and **jSonStr02** is a JSON string.

Run the scripts

I encourage you to copy the code from Listing 16 , Listing 17 , Listing 18 , and Listing 13 . Load the code into your favorite browser and observe the output. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Debugging JavaScript

Finding and correcting errors in your JavaScript code can be difficult. Sometimes when your JavaScript isn't working properly, opening the Firefox Web Console will provide useful diagnostic information about the error.

To open the Web Console in Firefox version 26,

- Select Tools
- Hover on Web Developer
- Select Web Console

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Json0130: JSON and JavaScript
- File: Json0130.htm
- Published: 02/02/14
- Revised: 02/08/16

Note: Disclaimers:
Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.
I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module.
In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.
Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

Complete script listings

Listing 16 . Json0130a.htm.

```
<!--01/26/14
Illustrates the difference between the syntax of a
JavaScript object and a JSON string that looks a lot like
a JavaScript object.

Must parse the JSON string to turn it into a JavaScript
object before processing it using JavaScript.

Uses JSON.stringify to produce a JSON string from a
```

```
JavaScript object.

Uses JSON.parse to produce a JavaScript object from
a JSON string.-->

<!DOCTYPE html>
<html>
  <head>
      <title>ParseJSON01</title>
  </head>
  <body>
    <script>
      //Define a function
      function aMethod(){return true;};
      document.write("Create a JavaScript object.");
      var obj01 = {name:"Bill",age:31,method:aMethod};
      document.write(
           "<br/>Unsuccessful attempt to display object.");
      document.write("<br/>" + obj01);
      document.write("<br/>Display keys in object: ");
      for (var key in obj01) {
        if (obj01.hasOwnProperty(key)) {
          document.write(key + " ");
        }//end if
      }//end for loop
      document.write("<br/>Display values in object: ");
      document.write(obj01.name + ", "
                     + obj01.age + ", " + obj01.method());

      document.write("<br/>Transform JavaScript object " +
                                      "into a JSON string.");
      // Note that the method does not become part of the
      // JSON string.
      var jsonstring = JSON.stringify(obj01);
      document.write(
                "<br/>Display JSON string " + jsonstring );
      document.write("<br/>Unsuccessful attempt to " +
                                     "access name and age.");
      document.write("<br/>" + jsonstring.name + ", "
                                          + jsonstring.age);

      document.write("<br/>Transform the JSON string " +
                                "into a JavaScript object.");
      var obj02 = JSON.parse(jsonstring);
      document.write("<br/>Display values in object: ");
```

```
      document.write(obj02.name + ", " + obj02.age);

      document.write("<br/>Display keys in object: ");;
      // Note that it does not contain the method from the
      // original JavaScript object.
      for (var key in obj02) {
        if (obj02.hasOwnProperty(key)) {
          document.write(key + " ");
        }//end if
      }//end for loop

    </script>
  </body>
</html>
```

Listing 17 . Json0130b.htm.

```
<!--01/26/14-----------------------------------------
-//
Illustrates the difference between the syntax of a
JavaScript object a JSON string that looks a lot like a
JavaScript object where each involves array data. Must
parse the JSON text to turn it into a JavaScript object
before processing it using JavaScript. -->
<!DOCTYPE html>
<html>
  <head>
    <title>ParseJSON02</title>
  </head>
  <body>
    <script>
      document.write("Create a JavaScript object " +
                                    "involving array
data.");
      var obj01 = {friends:[
              {name:"Bill",age:31},
{name:"Jill",age:40}]};
      document.write("<br/>Display values in object.");
```

Listing 17 . Json0130b.htm.

```
      document.write("<br/>" + obj01.friends[0].name +
                               ":" +
obj01.friends[0].age);
      document.write("<br/>" + obj01.friends[1].name +
                               ":" +
obj01.friends[1].age);

      document.write(
          "<br/>Transform the object into a JSON
string.");
      var jsontext = JSON.stringify(obj01);
      document.write("<br/>Display JSON string.");
      document.write("<br/>" + jsontext);
      document.write("<br/>Transform the JSON string " +
                              "into a JavaScript
object.");
      var obj02 = JSON.parse(jsontext);
      document.write("<br/>Display values in object.");
      document.write("<br/>" + obj02.friends[0].name
                              + ":" +
obj02.friends[0].age);
      document.write("<br/>" + obj02.friends[1].name
                              + ":" +
obj02.friends[1].age);

    </script>
  </body>
</html>
```

Listing 18 . Json0130d.htm.

Listing 18 . Json0130d.htm.

```
<!DOCTYPE html>
<html>
<body>

  <script>
    var jScrObj01 =  {"name":"John","age":33};
    var jSonStr01 = '{"name":"Bill","age":33}';

    document.write("<br/>1. " + jScrObj01.name);
    document.write("<br/>2. " + jSonStr01.name);

    document.write("<br/>3. " + jScrObj01);
    document.write("<br/>4. " + jSonStr01);

    var jScrObj02 = JSON.parse(jSonStr01);
    var jSonStr02 = JSON.stringify(jScrObj01);

    document.write("<br/>5. " + jScrObj02.name);
    document.write("<br/>6. " + jSonStr02.name);

    document.write("<br/>7. " + jScrObj02);
    document.write("<br/>8. " + jSonStr02);
  </script>

</body>
</html>
```

-end-

Json0140-Calling External JavaScript Functions
Learn how to write and call external JavaScript functions when working with JSON strings.

Table of Contents

Preface

This module is one in a collection of modules designed for teaching **ITSE 1356 Introduction to XML** at Austin Community College in Austin, TX.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the Figures and the Listings while you are reading about them.

Figures

Listings

Preview

Previous modules in this collection have shown you how to transform JavaScript objects into JSON strings, and how to transform JSON strings into JavaScript objects. Those program placed all of the JavaScript code in an HTML file. Sometimes you will need to break some of the JavaScript code out into a separate file. You will learn how to do that in this module.

Discussion and sample code

This module will show you how to create an HTML file *(**Asg.htm**)* and a separate JavaScript file *(**Asg.js**)* containing functions that are called by JavaScript code in the HTML file. The JavaScript file will contain two functions:

- A function named **objToStr** receives a JavaScript object as an incoming parameter and returns a JSON string that represents the object.
- A function named **strToObj** receives a JSON string as an incoming parameter and returns a JavaScript object that represents the string.

A complete listing of the HTML file is provided in Listing 4 near the end of the lesson. A complete listing of the JavaScript file is provided in Listing 5 near the end of the lesson.

Program output

Figure 1 shows the text that appears in a browser window when the files named **Asg.htm** and **Asg.js** are stored in the same folder and the file named **Asg.htm** is opened in a browser. Keep this figure handy so that you can refer to it while viewing the code.

Figure 1 . Program output.

```
Display keys in object: name age
Display values in object: Bill, 31
Transform JavaScript object into a JSON
string.

In objToStr

Display JSON string {"name":"Bill","age":31}
Transform the JSON string into a JavaScript
object.

In strToObj

Display values in object: Bill, 31
Display keys in object: name age
```

Will explain in fragments

As is often the case, I will explain this program in fragments. The beginning of the file named **Asg.htm** is shown in Listing 1 .

Listing 1 . Beginning of the file named Asg.htm.

```
<!DOCTYPE html>
<html>
    <head>
        <title>ITSE 1356</title>
        <script type='text/javascript'
src='Asg.js'></script>
        </head>
        <body>
           <script>
              var obj01 =
{name:"Bill",age:31};
              document.write("<br/>Display
keys in object: ");
              for (var key in obj01) {
                if (obj01.hasOwnProperty(key))
{
                  document.write(key + " ");
                }//end if
              }//end for loop
              document.write("<br/>Display
values in object: ");
              document.write(obj01.name + ", "
+ obj01.age);
```

The only new code in Listing 1 is the fifth line, which declares that a file named **Asg.js** contains JavaScript code that may be called by JavaScript code in the HTML file. Otherwise, the code in Listing 1 simply creates a JavaScript object and displays its contents. You have seen code like that before.

Call a function named objToStr

The second statement in Listing 2 is new to this module. This statement calls a JavaScript function named **objToStr** passing the JavaScript object as a parameter.

Listing 2 . Call a function named objToStr.

```
                document.write("<br/>Transform
JavaScript object " +

"into a JSON string.");
                var jsonstring =
objToStr(obj01);
```

When the JavaScript interpreter encounters this statement, it searches the current file *(**Asg.htm**)* for a function having that name. When it doesn't find the function in the current file, it searches the file named **Asg.js** , which was declared in Listing 1 . It finds the function in that file *(see Listing 5)* and executes it.

The function named objToStr

The code in the function named **objToStr** is straightforward. You have seen code like this before. After announcing that it has been called, the **objToStr** function calls the **JSON.stringify** method to create and return a JSON string based on the contents of the JavaScript object.

Call a function named strToObj

Returning to the code in the file named **Asg.htm** , the third statement in Listing 3 calls a function named **strToObj** passing the JSON string as a parameter.

Listing 3 . Call a function named strToObj.

```
              document.write(
                        "<br/>Display JSON
string " + jsonstring );

              document.write("<br/>Transform
the JSON string " +
                                      "into a
JavaScript object.");
              var obj02 =
strToObj(jsonstring);
```

The function named **strToObj** is also shown in Listing 5 . The code in this function is also straightforward and you have seen code like this before.

The **strToObj** function calls the **JSON.parse** method to create and return a JavaScript object based on the contents of the JSON string.

Following that, the code in the file named **Asg.htm** simply displays the contents of the JavaScript object. You can view that code in Listing 4 .

Run the program

I encourage you to copy the code from Listing 4 and Listing 5 and execute it by opening the HTML file in your browser. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Json0140-External JavaScript Functions
- File: Json0140.htm
- Published: 11/02/14

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

Complete program listings

Complete listings of the HTML file and the JavaScript file are shown below.

Listing 4 . The HTML file named Asg.htm.

```
<!DOCTYPE html>
<html>
    <head>
        <title>ITSE 1356</title>
        <script type='text/javascript'
src='Asg.js'></script>
        </head>
        <body>
           <script>
              var obj01 =
{name:"Bill",age:31};
              document.write("<br/>Display
```

Listing 4 . The HTML file named Asg.htm.

```
keys in object: ");
              for (var key in obj01) {
                if (obj01.hasOwnProperty(key))
{
                  document.write(key + " ");
                }//end if
              }//end for loop
              document.write("<br/>Display
values in object: ");
              document.write(obj01.name + ", "
+ obj01.age);

              document.write("<br/>Transform
JavaScript object " +

"into a JSON string.");
              var jsonstring =
objToStr(obj01);

              document.write(
                        "<br/>Display JSON
string " + jsonstring );

              document.write("<br/>Transform
the JSON string " +
                                        "into a
JavaScript object.");
              var obj02 =
strToObj(jsonstring);

              document.write("<br/>Display
values in object: ");
              document.write(obj02.name + ", "
+ obj02.age);
```

Listing 4 . The HTML file named Asg.htm.

```
               document.write("<br/>Display
keys in object: ");;
               for (var key in obj02) {
                 if (obj02.hasOwnProperty(key))
{
                   document.write(key + " ");
                 }//end if
               }//end for loop
            </script>
         </body>
</html>
```

Listing 5 . The JavaScript file named Asg.js.

Listing 5 . The JavaScript file named Asg.js.

```
function objToStr(obj){
        document.write("<br/><br/>In
objToStr<br/>");
        var str = JSON.stringify(obj);
        return str;
}

function strToObj(str){
        document.write("<br/><br/>In
strToObj<br/>");
        var obj = JSON.parse(str);
        return obj;
}
```

-end-

XML - Namespaces - Flex 3
Learn enough about XML namespaces to be able to handle the namespace requirements of simple Flex projects. Also get your first taste of creating MXML files for Flex projects.

Note: Click Namespace01 to run this Flex program. *(Click the "Back" button in your browser to return to this page.)*

Table of Contents

Preface

General

This tutorial lesson is part of a series dedicated to programming with Adobe Flex.

Flex is a programming language based on XML. Therefore, in order to program with Flex, you must first understand XML. The lessons in this XML series provide a brief introduction to XML.

Viewing tip

I recommend that you open another copy of this document in a separate browser window and use the following links to easily find and view the figures and listings while you are reading about them.

Figures

Listings

Supplemental material

I recommend that you also study the other lessons in my extensive collection of online programming tutorials. You will find a consolidated index at www.DickBaldwin.com .

General background information

XML namespaces make it possible to combine two or more XML documents into a single XML document and to deal with problems that arise when the same name is used for an element in two or more of the XML documents.

Preview

The XML namespace topic is very broad. I won't attempt to cover the topic of namespaces in this lesson. Instead, I will illustrate and explain a somewhat restricted use of namespaces by explaining a Flex project that combines three XML documents with conflicting element names.

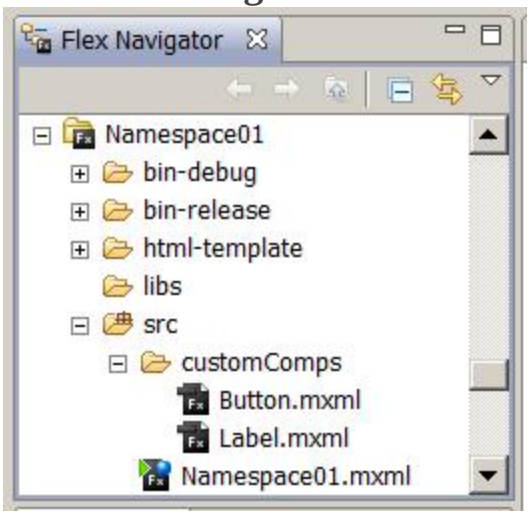
Before getting into the detailed code, I will show you a couple of images that I will be referring back to later.

The project file structure

When you create a Flex project, you create a directory tree on the disk to contain the files and folders that make up the project. The tree is rooted in a folder that has the same name as the name that you give to the project. In this lesson, I will explain a Flex project named Namespace01.

Figure 1 shows the directory tree for this project. Figure 1 is a snapshot of the Flex Builder 3 Navigator panel.

Figure The directory tree for the project named Namespace01.



The directory tree for the project named Namespace01.

Lots of folders and files

The directory tree contains a relatively large number of folders and files. Fortunately, most of the folders and files are generated automatically by Flex Builder 3. As Flex programmers, we are primarily interested in the folders and files that are children of the folder named **src** .

For this project I had to create the following files and folders:

- The file named **Namespace01.mxml**
- The folder named **customComps**
- The file named **Button.mxml**
- The file named **Label.mxml**

Create a graphical user interface (GUI)

If you continue in your study of Flex programming, you will learn that one of the most important uses of Flex is to create a graphical user interface (GUI) to provide the user interface for programs that are written in the ActionScript programming language.

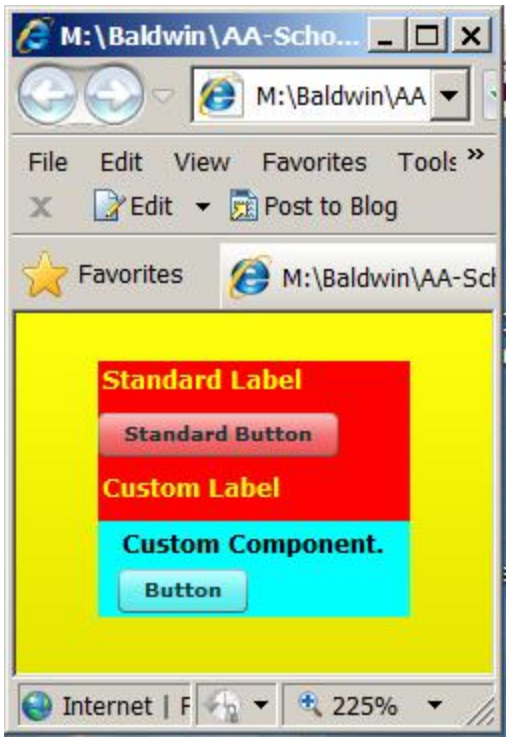
While you may not be familiar with the jargon term GUI, I know that you are familiar with the use of a GUI. The GUI is generally considered to consist of the buttons, menus, text fields, etc. that you interact with when you run a program on a modern desktop or laptop computer.

Two buttons, three labels, etc.

This project creates a GUI with two buttons and three labels in a **VBox** container with a red background as shown in Figure 2.

Program output at startup.

Figure



Program output at startup.

The buttons are not active

Nothing happens if you click the buttons in the GUI. The program isn't very interesting from an operational viewpoint. The thing that is interesting is that I purposely created name conflicts and resolved them through the use of XML namespaces. I will explain how I did that.

Two standard components

The size of the red **VBox** container was set to exactly match the size needed to contain the four components. The yellow label with the text **Standard Label** and the button labeled **Standard Button** are standard Flex components. As you will see shortly, they are created by creating instances of the standard Flex components named **Label** and **Button** .

A conflict with the name Label

The yellow label with the text **Custom Label** and the cyan rectangle containing a label and a button are both custom components. The yellow label in the middle was created by combining the XML file named **Namespace01.mxml** shown in Figure 1 with the file named **Label.mxml** , which is also shown in Figure 1. This created a name conflict because the name of the file, **Label** , is the same as the name of the yellow label at the top of the red **VBox** .

A conflict with the name Button

The cyan component containing the button and the label was created by combining the XML file named **Namespace01.mxml** shown in Figure 1 with the file named **Button.mxml** , which is also shown in Figure 1. This also created a name conflict because the name of the file, **Button** , is the same as the name of the standard button.

Start with a simple project

Now that you have the big picture in mind, it's time to drill down a little deeper and take a look at the three XML files used in this project. However, I'm going to start out with a very simple project and work my way up to the project that created the output shown in Figure 2.

Discussion and sample code

Two ways to create Flex projects

Flex projects can be created using nothing more than a text editor and a Flex software development kit (SDK) that is freely available from the Adobe website. However, to make the development of Flex projects a little easier, Adobe sells a product named Flex Builder 3 that includes the SDK along with a visual project editor. The project that I will explain in this lesson was created using Flex Builder 3.

> *(Note that as of June 2010, the product named Flex Builder 3 has been replaced by a product named Flash Builder 4 to*

> *accommodate the release of version 4 of Flex. I will be covering both Flex 3 and Flex 4 in this series of lessons.)*

Skeleton MXML code for a new Flex project

When you create a new Flex project in Flex Builder 3, a skeleton of the required MXML file is created for you. Listing 1 shows the contents of such a skeleton MXML file.

Example:
Skeleton MXML code for a new Flex project.

```
<?xml version="1.0" encoding="utf-8"?>

<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="absolute">

</mx:Application>
```

> *(Listing 1 shows the skeleton code for a Flex 3 project. The skeleton code for a Flex 4 project is somewhat more complicated.)*

The XML declaration

The line of code that begins with the left angle bracket followed by **?xml** is called the *XML declaration* . It tells what version of XML is being used and

also specifies the encoding scheme for the characters in the file. An XML declaration should be included at the beginning of every XML document.

The root element

The line of text that begins with a left angle bracket and **mx:Application** is the *start tag* for the *root element* in the XML document. *(The root element is always named **Application** in a Flex project.)* At this point in the course, you should be able to identify the end tag for the **Application** element.

All XML documents have a root element. All other elements must be nested in the root element.

Attributes of the root element

The start tag for the root element in Listing 1 includes two attributes:

- xmlns:mx
- layout

The layout attribute

The layout attribute specifies that components will be positioned on the computer screen using absolute coordinates. As you will see later, this is not what I want for my project so I will delete this attribute.

The namespace (xmlns) attribute

The more interesting attribute is the one named **xmlns** . The term **xmlns** is the required name for a namespace attribute. While it isn't necessary in general to include a namespace attribute in the root element, when a namespace attribute is included in the root element, it becomes the *default namespace* for the entire document.

Namespace is always required for a Flex project

It is always necessary to include the namespace attribute shown in Listing 1 in the main MXML document for a Flex 3 project. That is why Flex Builder 3 includes it in the skeleton code for the project.

> *(Flash Builder 4 includes different namespace attributes in the skeleton code for a Flex 4 project.)*

To make a long story short, the inclusion of the default namespace attribute shown in Listing 1 means that all elements with names that refer to components from the standard Flex 3 library of components must be prefixed with **mx** .

A viable Flex project

The code shown in Listing 1 is a viable Flex 3 project. You can compile it and run it. However, when you do, you won't see any output other than a blank area with a default gray background in your browser window. So far, the project doesn't contain any Flex components such as labels and buttons.

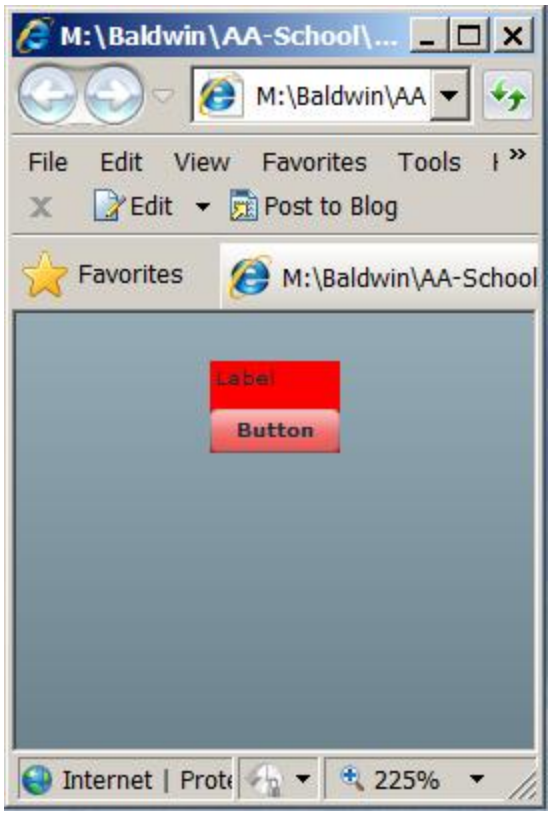
Screen output for upgraded Flex project

Now I am going to upgrade the project to add a **VBox** container with a red background to the **Application** element, and then add a **Label** and a **Button** to the **VBox** container element. I will also delete the **layout** attribute shown in Listing 1.

If you compile and run this project, you should see the output shown in Figure 3.

Screen output for upgraded Flex project.

Figure



Screen output for upgraded Flex project.

The modified MXML code

The modified MXML code is shown in Listing 2.

Example:
Upgraded Flex project code.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml">

  <!--Add a VBox container to the Application-->
```

```
  <mx:VBox backgroundColor="#FF0000">

    <!--Add a label and a button to the VBox-->
    <mx:Label text="Label"/>
    <mx:Button label="Button"/>

  </mx:VBox>

</mx:Application>
```

Comments

The lines of text in Listing 2 that begin with the left angle bracket followed by !-- are comments. The comment includes everything from the beginning to the -- followed by a right angle bracket. *(You cannot include -- in a comment.)*

You can include comments in an XML document to help explain the code, record the date, or for whatever purpose a comment may be appropriate. Comments are ignored by the Flex compiler and have no effect on the behavior of the program.

The Application element

The start and end tags for the **Application** element are the same as in Listing 1 except that I deleted the **layout** attribute.

The VBox element

At this point, you should have no difficulty identifying the start and end tags for the **VBox** element. Note that I included an attribute for the **VBox** element to cause the **backgroundColor** for the **VBox** to be red. *(I will leave it as an exercise for the student to research and determine how "#FF0000" represents the color red.)*

The VBox element

You should also note that the name of the **VBox** element is prefixed with **mx** , which is the default namespace for all Flex 3 components.

Finally, you should note that the **VBox** element is nested inside the Application element. We say that in this case, **VBox** is a child of **Application** .

The physical output on the screen

Physically, the **Application** element represents the browser window with the gray background shown in Figure 3. The **VBox** component with the red background is inside of or contained within the **Application** container in Figure 3.

The nesting structure that you give to the MXML code carries through to the physical arrangement of the corresponding components in the resulting GUI.

The Label and Button elements

The two lines of code that begin with a left angle bracket followed by **mx:Label** and **mx:Button** nest a **Label** element and a **Button** element inside the **VBox** element. Once again, this carries through to the output GUI shown in Figure 3 where the label and the button are contained in the red **VBox** component.

Note that as is the case with all standard Flex 3 components, the names of the **Label** and **Button** elements are prefixed with **mx** . *(A different prefix is used in Flex 4.)*

Properties of the Label and Button element

The **Label** element has an attribute that sets the **text** property to the text that you see in the label in Figure 3. Similarly, the **Button** element has an attribute that sets the button's **label** property to the text that you see on the face of the button in Figure 3.

Grouping components in containers

The Flex **VBox** component is a *container* . Its purpose is to serve as a container for other components. Flex 3 provides several other *container* components in addition to the **VBox** component. They are used to group components in the GUI in a way that helps the user navigate the GUI.

The Label and Button elements are empty

The **Label** and **Button** components are not containers. Therefore, they cannot contain other components in the GUI.

As a result, the **Label** and **Button** elements in Listing 2 are *empty* . They don't contain any other elements. However, even empty elements can and often do have attributes.

No end tags for Label and Button elements

Because the **Label** and **Button** elements are empty elements, they don't have end tags. Instead they have a slash character immediately to the left of the closing angle bracket on the start tag.

The final upgrade

Now I'm going to upgrade the project once again to produce the output GUI shown in Figure 2. In this upgrade, I will add a custom label and a custom button that create name conflicts. I will resolve the name conflict using namespaces.

Two custom components

The custom components are defined in the files named **Label.mxml** and **Button.mxml** shown in Figure 1. I will begin by discussing the MXML code for each of these custom components.

Listing 3 shows the MXML code for the custom label.

Example:
Contents of the file named Label.mxml.

```
<?xml version="1.0" encoding="utf-8"?>

<mx:VBox
xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Label
    text="Custom Label"
    color="#FFFF00"
    fontSize="12"
    fontWeight="bold"/>

</mx:VBox>
```

Won't discuss in detail

Since this lesson is mainly about using namespaces to resolve name conflicts, and is not about creating custom components, I'm not going to go into detail at this time about how to create custom components. Briefly, Listing 3 creates a custom component consisting of a label with yellow bold text and a font size of 12 points inside of a **VBox** container.

This custom component is shown as the label with the yellow text in the middle of Figure 2. *(I will cover the details of designing and creating custom components in a future lesson.)*

Contents of the file named Button.mxml

Listing 4 shows the MXML code for the custom button.

Example:

Contents of the file named Button.mxml.

```
<?xml version="1.0" encoding="utf-8"?>

<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
  backgroundColor="#00FFFF">

  <mx:Label
    text="Custom Component."
    color="#000000"
    fontSize="12" fontWeight="bold"/>

  <mx:Button
    label="Button"/>

</mx:VBox>
```

This custom component can also be viewed in Figure 2. Briefly it consists of a button and a label with black text in a small **VBox** container with a cyan background.

Contents of the file named NameSpace01.mxml

Listing 5 shows the contents of the file named **NameSpace01** , which is the main driver for the entire application.

Example:
Contents of the file named NameSpace01.mxml.

```
<?xml version="1.0"?>
<!--
Namespace01
Illustrates the use of namespaces to avoid name
```

```
conflicts.
-->
<mx:Application
xmlns:mx="http://www.adobe.com/2006/mxml"
                xmlns:MyComps="customComps.*"
                backgroundColor="#FFFF00">

  <!--Add a standard VBox container-->
  <mx:VBox backgroundColor="#FF0000">

    <mx:Label text="Standard Label"
      color="#FFFF00"
      fontSize="12"
      fontWeight="bold"/>
    <mx:Button label="Standard Button" />

    <MyComps:Label id="customLabel"/>
    <MyComps:Button id="customButton"/>
  </mx:VBox>

</mx:Application>
```

Resolving name conflicts using namespaces

This code adds two standard components named **Label** and **Button** and two custom components named **Label** and **Button** to a **VBox** container.

Because the custom components have the same names as the standard components, a name conflict arises. Listing 5 resolves the name conflict using namespaces.

The folder named customComps

As you may recall from Figure 1, the two files that represent the custom components are in a folder named **customComps** , which is a child of the

folder named **src** . Thus, the folder named **customComps** is a sibling of the file named **NameSpace01.mxml** .

A new namespace attribute

The code that begins with **xmlns:MyComps** in Listing 5 is a new attribute for the **Application** element. This attribute establishes a new namespace with the prefix **MyComps** . The namespace prefix points to all of the files in the folder named **customComps** .

The name **customComps** identifies the folder and the period and asterisk following the name mean that all of the files in the folder are part of the new namespace.

Must use the new prefix

In order to include the custom components defined by these files in the MXML document, elements named after these components must be prefixed with **MyComps** .

Using the mx prefix for standard Flex components

The code that begins with **mx** in Listing 5 uses the standard **mx** namespace prefix to add a standard **Label** component and a standard **Button** component to the **VBox** container with a red background. You should have no difficulty identifying the start and end tags for the **VBox** element. Attributes are applied to the **mx:Label** element to set the color, size, and weight of the text in the label.

Using the MyComps prefix for custom components

The two lines of code that begin with a left angle bracket and the word **MyComps** use the new **MyComps** namespace prefix to add a custom **Label** component and a custom **Button** component to the same **VBox** container.

Resolving the name conflict

Because the elements for standard components are prefixed with **mx** and the elements for custom components are prefixed with **MyComps** , the compiler is able to distinguish between them and to resolve the name conflicts.

The final result is the GUI shown in Figure 2.

Resources

I will publish a list containing links to Flex resources as a separate document. Search for Flex Resources in the Connexions search box.

Miscellaneous

This section contains a variety of miscellaneous materials.

Note: Housekeeping material

- Module name: XML - Namespaces - Flex 3
- Files:
    - Flex0086\Flex0086.htm
    - Flex0086\Connexions\FlexXhtml0086.htm

-end-

XML - Namespaces - Flex 4
Learn about namespace differences and some of the other differences between Flex 3 and Flex 4.

Note: Click Namespace02 to run the Flex program from this lesson. *(Click the "Back" button in your browser to return to this page.)*

Table of Contents

Preface

General

This tutorial lesson is part of a series of lessons dedicated to programming using Adobe Flex.

Note: The material in these lessons is based on Flex version 3 and Flex version 4. A distinction between the two will usually be made in those situations where that distinction is important.

A previous lesson in this series titled XML - Namespaces - Flex 3 concentrated on teaching the XML concept of *namespaces* and illustrated the concept using a program written in Flex version 3.

Differences in namespaces between Flex 3 and Flex 4

Some of the first things that one is likely to notice when comparing Flex version 3 to Flex version 4 are some obvious differences in the use of namespaces. Therefore, this is an opportune place in the series to introduce Flex version 4 and to explain some of the differences between the two versions.

Viewing tip

I recommend that you open another copy of this document in a separate browser window and use the following links to easily find and view the figures and listings while you are reading about them.

Figures

- Figure 1 . Output from Namespace01.
- Figure 2 . Output from Namespace02.
- Figure 3 . Project tree for the project named Namespace02.

Listings

Supplemental material

I recommend that you also study the other lessons in my extensive collection of online programming tutorials. You will find a consolidated index at www.DickBaldwin.com .

General background information

Historical perspective

Adobe's Flex is an XML-based programming language that is used to create programs that execute in the Adobe Flash Player .

Teaching XML using Flex

In the Spring semester of 2010, I introduced Adobe's *Flex version 3* and the *Flex Builder 3 IDE* into a course named **Introduction to XML** that I had been teaching for several years at Austin Community College in Austin, TX. The concept of using Flex as the programming vehicle to teach XML was well received by the students.

During that same semester, Adobe released *Flex version 4* and replaced *Flex Builder 3* with a new IDE named *Flash Builder 4* . The new IDE supports both Flex 3 and Flex 4.

A fortunate circumstance

This is a fortunate circumstance insofar as the concept of using Flex to teach XML is concerned. Flex 4 is similar to, but very different from, and somewhat more complicated than Flex 3. The availability of the two versions of Flex makes it possible for the students to gain experience with two similar but different flavors of XML, both supported by the same IDE and both supported by similarly formatted documentation.

What is Flex?

As mentioned above, Flex is an XML-based programming language that is used to create programs that execute in Adobe's Flash Player. In order to understand Flex, and particularly the differences between Flex 3 and Flex 4, we need to start with the Flash Player and work backwards to Flex.

What is the Flash Player?

According to the Flash Player website:

> Adobe Flash Player is a cross-platform browser-based application runtime that delivers uncompromised viewing of expressive applications, content, and videos across screens and browsers. Flash Player delivers breakthrough web experiences to over 98% of Internet users.

Flash Player is widely available

Many of the popular websites that people frequently visit require that the Flash Player be installed on the local computer in order to view the material

on the website.

Typically if you visit a website that requires the Flash Player and you don't have it installed on your computer, you will be guided through the installation process. Therefore, a very large percentage of computers already have the Flash Player installed.

An execution engine

In short, the Flash Player is an execution engine that is used to execute or *play* programs that are written in the *ActionScript* programming language. *(See Baldwin's ActionScript programming website .)*

What is ActionScript?

According to the ActionScript Technology Center,

> "Adobe ActionScript is the programming language of the Adobe Flash Platform. Originally developed as a way for developers to program interactivity, ActionScript enables efficient programming of Adobe Flash Platform applications for everything from simple animations to complex, data-rich, interactive application interfaces.

> First introduced in Flash Player 9, ActionScript 3.0 is an object-oriented programming (OOP) language based on ECMAScript -- the same standard that is the basis for JavaScript -- and provides incredible gains in runtime performance and developer productivity."

What is the Adobe Flash Platform?

According to Adobe Flash Platform ,

> "The Adobe Flash Platform is an integrated set of technologies surrounded by an established ecosystem of support programs, business partners, and enthusiastic user communities. Together, they provide everything you need to create and deliver the most compelling applications, content, and video to the widest possible audience."

The primary delivery mechanisms for applications built with the Adobe Flash Platform are the Adobe Flash Player and Adobe Air .

What is Adobe Air?

According to Adobe Air ,

> "The Adobe AIR runtime lets developers use proven web technologies to build rich Internet applications that run outside the browser on multiple operating systems."

Once again, what is Flex?

Flex is an *XML-based* programming language that can be used to create *ActionScript* programs for execution in the *Flash Player* . When you compile a Flex project, it is first converted into an ActionScript program and the ActionScript program is compiled into a form suitable for execution by the Flash Player.

According to The Adobe Flash Builder 4 and Flex 4 Bible by David Gassner,

> When you compile a Flex application, your MXML code is rewritten in the background into pure ActionScript 3. MXML can be described as a "convenience language" for ActionScript 3 that

> makes it easier and faster to write your applications that if you had to code completely in ActionScript.

Easier and faster is debatable

In my opinion, as a person with many years of object-oriented programming experience, it is debatable whether coding ActionScript programs in Flex is *easier and faster* than coding them in pure ActionScript. Any program that can be coded in Flex can also be coded in pure ActionScript, but the reverse is not true.

XML, not ActionScript

In any event, the purpose of the lessons in this series is to teach XML and not to teach ActionScript programming. *(ActionScript OOP is a different course that I teach at the college.)* Therefore, insofar as practical, the lessons in this series will concentrate on Flex programming and not on ActionScript programming.

However, to understand the differences between Flex 3 and Flex 4, it will sometimes be necessary to refer to ActionScript, particularly insofar as the documentation is concerned.

Preview

Run the Flex program named Namespace02

If you have the Flash Player plug-in *(version 10 or later)* installed in your browser, click here to run the program that I will explain in this lesson.

If you don't have the proper Flash Player installed, you should be notified of that fact and given an opportunity to download and install the Flash Player plug-in program.

Namespaces is an XML concept

The concept of **namespaces** is an XML concept. It is not a concept that is exclusive to Flex. However, because Flex is an XML-based programming language, Flex makes heavy use of namespaces.

I explained the concept of XML namespaces in the earlier lesson titled XML - Namespaces - Flex 3 . I also presented and explained a relatively simple Flex program that illustrated the use of XML namespaces to resolve name conflicts.

Some of the differences between Flex 3 and Flex 4

In this lesson, I will present a somewhat broader view of namespaces and will also present and explain a program that illustrates some of the differences between Flex 3 and Flex 4.

In explaining the differences between Flex 3 and Flex 4, I will need to dig a little more deeply into the Flex programming language than was the case in the earlier lesson.

The program that I explained in the earlier lesson was written exclusively using Flex 3. The program that I will explain in this lesson was written exclusively in Flex 4. The new Flex 4 program approximates the look and feel of the Flex 3 program from the earlier lesson.

Discussion and sample code

Two ways to create Flex projects

As I explained in the earlier lesson, Flex projects can be created using nothing more than a text editor and a Flex software development kit *(SDK)* that is freely available from the Adobe website. However, to make the development of Flex projects a little easier, Adobe previously sold a product named *Flex Builder 3* and now sells a replacement product named *Flash Builder 4* , which includes the Flex 3 and Flex 4 SDKs along with a visual project editor.

The project that I explained in the earlier lesson was created using Flex Builder 3. The project that I will explain in this lesson was created using

Flash Builder 4.

Free for educational use

As of June 2010, Adobe provides free copies of Adobe Flash Builder 4 Standard to:

- Students, faculty and staff of eligible educational institutions
- Software developers who are affected by the current economic condition and are currently unemployed
- Event attendees who receive a special promotional code at their event

Skeleton mxml code and namespaces

Skeleton mxml code for a new Flex 3 project

When you create a new Flex 3 project in Flex Builder 3 or Flash Builder 4, a skeleton of the required mxml file is created for you. Listing 1 shows the contents of such a skeleton mxml file for a Flex 3 project.

Example:
Skeleton mxml code for a new Flex 3 project.

```
<?xml version="1.0" encoding="utf-8"?>

<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="absolute">

</mx:Application>
```

Created using Flex Builder 3

The skeleton code shown in Listing 1 was created using Flex Builder 3, but the skeleton code for a Flex 3 project is essentially the same regardless of whether it is created using Flex Builder 3 or Flash Builder 4. *(Flash Builder 4 inserts a couple of relatively insignificant size attributes that are not inserted by Flex Builder 3.)*

The namespace (xmlns) attribute

In the earlier lesson, I explained the concept of the *root element* , and I explained that the term **xmlns** is the required name for a *namespace* attribute. *(This is true for XML in general and not just for Flex mxml.)* While it isn't necessary in general to include a namespace attribute in the root element, when a namespace attribute is included in the root element, it becomes the *default namespace* for the entire document.

Namespace is always required for a Flex project

Even though it isn't necessary to include a namespace attribute in the root element of a general XML document, it is always necessary to include the namespace attribute shown in Listing 1 in the root element of the main mxml document for a Flex 3 project. That is why Flex Builder 3 includes it in the skeleton code for the project.

What does this mean?

The inclusion of the default namespace attribute shown in Listing 1 means that all elements with names that refer to components from the standard Flex 3 library of components must be prefixed with **"mx:"** .

Skeleton mxml code for a new Flex 4 project

As with a Flex 3 project, when you create a new Flex 4 project in Flash Builder 4, a skeleton of the mxml file is created for you. Listing 2 shows the contents of such a skeleton mxml file for a Flex 4 project.

Example:

Skeleton mxml code for a new Flex 4 project.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application
xmlns:fx="http://ns.adobe.com/mxml/2009"

xmlns:s="library://ns.adobe.com/flex/spark"

xmlns:mx="library://ns.adobe.com/flex/mx"
               minWidth="955"
               minHeight="600">
  <fx:Declarations>
    <!-- Place non-visual elements (e.g.,
services,
    value objects) here -->
  </fx:Declarations>
</s:Application>
```

More namespace attributes in the root element

If you compare Listing 2 with Listing 1, you will see that the namespace attributes in Listing 2 are different from those in Listing 1, and there are more of them in Listing 2.

Listing 1 has only one namespace attribute while Listing 2 has three namespace attributes.

Mix or match Flex components

You can use Flash Builder 4 to create projects that

- use Flex 3 exclusively
- use Flex 4 exclusively
- use a mixture of the two

Must specify compiler version for project

When you create a new project in Flash Builder 4, you must specify whether the project is to be compiled using the Flex 3 compiler or the Flex 4 compiler.

Different versions of the skeleton code

If you specify the Flex 3 compiler, the skeleton code will look like Listing 1 *(with a couple of additional sizing attributes)* . For that case, you must use Flex 3 components exclusively.

If you specify the Flex 4 compiler, the skeleton code will look like Listing 2. In that case, you can use Flex 3 components, Flex 4 components, or a mixture of the two.

What do these namespace attributes mean?

Building on what I explained earlier, the inclusion of the namespace attributes with the name **"mx"** in Listing 1 and Listing 2 means that all elements with names that refer to components from the Flex 3 library of components must be prefixed with **"mx:"** . *(You will see examples of this in code fragments later in this lesson.)*

The inclusion of the namespace attribute with the name **"s"** in Listing 2 means that all elements with names that refer to the new components from the Flex 4 library of components must be prefixed with **"s:"** . *(You will also see examples of this in code fragments later in this lesson.)*

Resolution of duplicate names

The Flex 3 library and the Flex 4 library contain many components with the same names, such as **Label** and **Button** . Therefore, the name of the component alone is not sufficient to identify which of two components having the same name is to be used at a particular location in the program. The **"mx:"** prefix and the **"s:"** prefix are the mechanisms by which you identify the correct component to the compiler.

Note: For those with knowledge of ActionScript or Java programming, this is analogous to using a package name to identify a class in those programming languages.

You can read more on the topic of required namespaces here .

The sample program named Namespace02

Figure 1 shows the output from the Flex 3 program named **Namespace01** that I explained in the earlier lesson on this topic.

Output from Namespace01.

Figure

Output from Namespace01.

Figure 2 shows the output from the Flex 4 program named **Namespace02** that I will explain in this lesson.

Output from Namespace02.

Figure

Output from Namespace02.

Mostly default look and feel

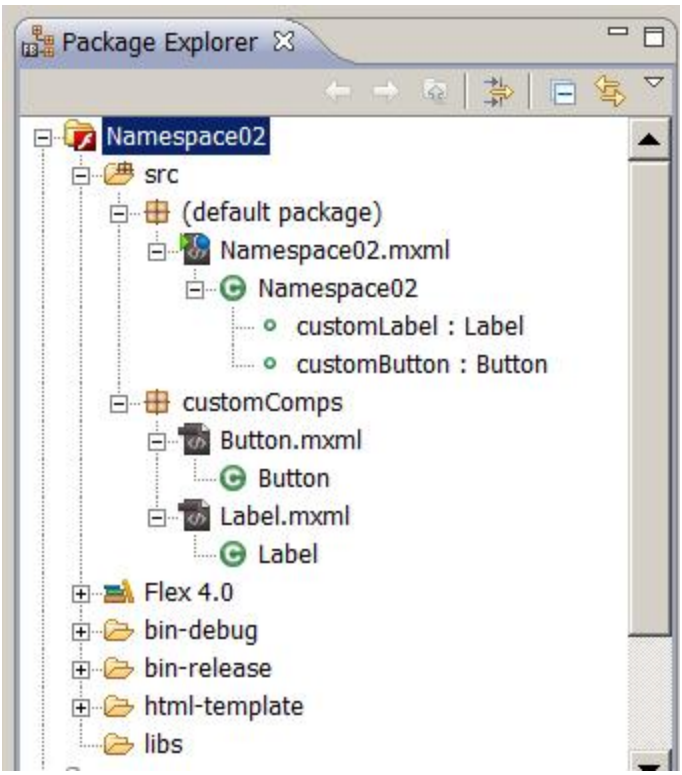
In both programs, the top portion of the output was purposely colored red and the bottom portion was purposely colored cyan. Otherwise, the colors, sizes, positions, and shapes of the components in both programs were allowed to take on default values.

The project tree for the project named Namespace02

The project tree for the Flex 4 project named **Namespace02** is shown in Figure 3.

Project tree for the project named Namespace02.

Figure



Project tree for the project named Namespace02.

A comparable image for the Flex 3 project named **Namespace01** was provided in the earlier lesson. If you compare the two, you will see that more information is displayed in the project tree for the Flex 4 project in Figure 3.

Major items of interest

For purposes of this lesson, we will be primarily interested in the following items showing in Figure 3. Those are the items that I had to create in order to create the project.

- The file named Namespace02.mxml
- The folder named customComps
- The file named Button.mxml
- The file named Label.mxml

Two buttons, three labels, etc.

As I explained in the earlier lesson, the project named **Namespace01** creates a GUI with two buttons and three labels in **VBox** containers with red and cyan backgrounds as shown in Figure 1.

All are mx components

Because that project was created exclusively using Flex 3, all of the components shown in Figure 1 are Flex 3 components. I will sometimes refer to them as **"mx"** components because of the name of the namespace attribute shown in Listing 1.

No VBox components in Namespace02

Because the Flex 4 program named **Namespace02** was intended to replicate **Namespace01** , it also contains two buttons and three labels. However, as you will see later, they are not in **VBox** containers because there is no **VBox** container in Flex 4. Instead, they are in containers named **Group** and **VGroup** .

All are Spark components

Because **Namespace02** was created exclusively using Flex 4, all of the components are Flex 4 components. I will sometimes refer to them as **"Spark"** components on the basis of the last word in the value of the namespace attribute named **"s"** in Listing 2.

Note: The names "mx" and "Spark" actually derive from ActionScript package names, but an explanation of that is beyond the scope of this lesson.

The main mxml file for Namespace01

Listing 3 shows the code in the main mxml file for the Flex 3 project named **Namespace01** .

Example:
The main mxml file for Namespace01.

```
<?xml version="1.0"?>
<!--
Namespace01
Illustrates the use of namespaces to avoid name
conflicts.
-->
<mx:Application
xmlns:mx="http://www.adobe.com/2006/mxml"
                xmlns:MyComps="customComps.*"
                backgroundColor="#FFFF00">

  <!--Add a standard VBox container-->
  <mx:VBox backgroundColor="#FF0000">

    <mx:Label text="Standard Label"
      color="#FFFF00"
      fontSize="12"
      fontWeight="bold"/>
    <mx:Button label="Standard Button" />

    <MyComps:Label id="customLabel"/>
    <MyComps:Button id="customButton"/>
  </mx:VBox>

</mx:Application>
```

The Application element

I will explain **mxml** syntax in more detail in future lessons, so I'm not going to go into syntax issues at this point in time. Suffice it to say that the **Application** element in Listing 3 represents the entire program. The behavior as well as the look and feel of the program is defined by the attributes and the content of the **Application** element. Everything in the program is part of the attributes or the content of the **Application** element.

An mx:VBox element

From what you already know about XML, you can see that an element named **mx:VBox** is part of the content of the **Application** element. Very briefly, in Flex, an **mx:VBox** element is a container element that can contain other elements. Couched in visual terms such as Figure 1, an **mx:VBox** object can contain other components such as labels and buttons.

Note that the **mx:VBox** element name has an **mx** prefix, meaning that it represents a component from the Flex 3 library as explained earlier.

The backgroundColor attribute of the mx:VBox element

Also note that the **mx:VBox** element has an attribute named **backgroundColor** with a value of **"#FF0000"** . In a future lesson, I will explain that this is a hexadecimal value that represents the color red at maximum intensity. This attribute produces the red background color that you see in the upper portion of Figure 1.

Note: The lower portion of Figure 1 also has a red background color, but it is covered by another mx:VBox element with an opaque cyan background color.

The backgroundColor attribute for the Application element

While we are discussing background colors, it is also worth mentioning that the **application** element has an attribute named **backgroundColor** with a

value of **"#FFFF00"** . This is the hexadecimal value for yellow and causes the background color of the entire Flash Player window to be yellow.

Contents of the mx:VBox element

The **mx:VBox** element contains the following four elements:

- mx:Label
- mx:Button
- MyComps:Label
- MyComps:Button

In the earlier lesson, I explained that the first two of these four elements represent components from the standard Flex 3 library. *(Hence the "mx:" prefix.)* The last two represent custom components that were constructed using components from the standard Flex 3 library.

The MyComps:Button element

If you go back to the earlier lesson and examine the code for the custom component named **MyComps:Button** , you will see that it has an **mx:Label** and an **mx:Button** in an **mx:VBox** container with a backgroundColor value of **"#00FFFF"** *(cyan)* . This produces the cyan rectangle containing the label and the button in the bottom portion of Figure 1.

The main mxml file for Namespace02

The main mxml file for the Flex 4 project named **Namespace02** is shown in Listing 4.

Example:
The main mxml file for Namespace02.

```
<?xml version="1.0" encoding="utf-8"?>

<!--File: Namespace02.mxml
```

```
This is a Flex 4 version of the Flex 3 program
named Namespace01-->

<!--Declare a namespace as the folder named
customComps,
which contains a custom label component and a
second
custom component consisting of a Spark Label and a
Spark Button. Then declare the three namespaces
required
by Flex 4. Finally cause the background to be
yellow.-->
<s:Application xmlns:MyComps="customComps.*"

xmlns:fx="http://ns.adobe.com/mxml/2009"

xmlns:s="library://ns.adobe.com/flex/spark"

xmlns:mx="library://ns.adobe.com/flex/mx"
               backgroundColor="0xFFFF00">

  <!--Put a Spark Label and a Spark Button along
with two
  custom components in a Spark Group with a red
background
  color.-->
  <s:Group horizontalCenter="0"
verticalCenter="0">

    <!--Create a red rectangle to serve as the
background
    color for the Group-->
    <s:Rect width="100%" height="100%">
      <s:fill>
        <s:SolidColor color="0xFF0000" />
      </s:fill>
    </s:Rect>
```

```
    <!--Add a Spark VGroup to contain the
components and
    cause them to be laid out vertically.-->
    <s:VGroup>
      <!--Add two Spark components to the VGroup--
>
      <s:Label text="Spark Label"
               color="#FFFF00"
               fontSize="12"
               fontWeight="bold"/>
      <s:Button label="Spark Button" />

      <!--Add two custom components to the VGroup-
->
      <MyComps:Label id="customLabel"/>
      <MyComps:Button id="customButton"/>
    </s:VGroup>

  </s:Group>

</s:Application>
```

Lots of comments

As you can see, I included lots of comments in Listing 4 in an attempt to make it as self-explanatory as possible.

In this lesson, I will concentrate on the differences between this Flex 4 project and the Flex 3 project named **Namespace01** that arise from creating the two projects using different versions of Flex.

Order of attributes is not important

Let me begin by explaining that in XML, the order in which you write the attributes for an element doesn't matter so long as they are all there with the correct syntax, the correct names, and the correct values.

More and different namespace attributes

As I explained earlier, a Flex 4 project often has three required namespace attributes and almost always has two. *(Because I didn't use any mx components in this program, I could have removed the namespace attribute named mx from Listing 4.)*

Other than the namespace attributes, the **application** element in Listing 4 has the same attribute names and values as the **application** element in Listing 3.

No VBox element in Namespace02

The next thing to notice is that there is no **mx:VBox** element in Listing 4. Instead, there is an **s:Group** element *(a Flex 4 Spark component)* that replaces the **mx:VBox** element and serves as a container for the labels and the buttons.

No backgroundColor attribute

The **s:Group** element has two positioning attributes that cause it to appear in the center of the Flash Player window, but it does not have an attribute named **backgroundColor** . Like many of the Spark components, and unlike many of the mx components, the **s:Group** element does not have built-in attributes that are used to control its appearance. Instead, other ways must be found to control the appearance of many Spark components.

A red rectangle

In this case, Listing 4 causes the **s:Group** element to appear to have a red background by causing it to contain a red rectangle of exactly the right dimensions to completely fill the **s:Group** element. This produces the red background color in the upper portion of Figure 2.

Note: As with Figure 1, the lower portion of Figure 2 also has a red background color, but it is covered by another smaller rectangle with an opaque cyan color.

Add an s:VGroup container

Then Listing 4 adds a Spark **s:VGroup** container to serve essentially the same purpose as the **mx:VBox** container in Listing 3 *(except that it doesn't control the red background color)* . The following elements are added to the **s:VGroup** element in Listing 4 In a manner very similar to Listing 3:

- s:Label
- s:Button
- MyComps:Label
- MyComps:Button

The first two elements in the above list are Spark elements having similar characteristics to the mx elements having the same names.

The last two elements in the above list are custom components having similar characteristics to the custom components having the same names in the earlier program.

Contents of the file named Label.mxml

The contents of the custom component file named **Label.mxml** are shown in Listing 5.

Example:
Contents of the file named Label.mxml.

```
<?xml version="1.0" encoding="utf-8"?>

<!--Create a custom label by putting a Spark Label
```

```
in
a Spark Group-->
<s:Group xmlns:MyComps="customComps.*"
         xmlns:fx="http://ns.adobe.com/mxml/2009"

xmlns:s="library://ns.adobe.com/flex/spark"

xmlns:mx="library://ns.adobe.com/flex/mx">

  <s:Label
    text="Custom Label"
    color="#FFFF00"
    fontSize="12"
    fontWeight="bold"/>

</s:Group>
```

Contents of the file named Button.mxml

The contents of the custom component file named **Button.mxml** are shown in Listing 6.

Example:
Contents of the file named Button.mxml.

```
<?xml version="1.0" encoding="utf-8"?>

<!--Create a custom component by putting a Spark
Label
and a Spark Button in a Spark VGroup inside of a
Spark
Group with a Cyan background color.-->
<s:Group  xmlns:MyComps="customComps.*"
```

```
          xmlns:fx="http://ns.adobe.com/mxml/2009"

xmlns:s="library://ns.adobe.com/flex/spark"

xmlns:mx="library://ns.adobe.com/flex/mx">

  <!--Fill the entire group with a cyan rectangle-
->
  <s:Rect width="100%" height="100%">
    <s:fill>
      <s:SolidColor color="0x00FFFF" />
    </s:fill>
  </s:Rect>

  <!--Put a Spark VGroup in the Group and put a
Spark
  Label and a Spark Button in the VGroup-->
  <s:VGroup>
    <s:Label
      text="Custom Component."
      color="#000000"
      fontSize="12" fontWeight="bold"/>

    <s:Button
      label="Button"/>
  </s:VGroup>
</s:Group>
```

No further explanation needed

Assuming that you understand the contents of the files named **Label.mxml** and **Button.mxml** in the Flex 3 program in the earlier lesson, and assuming that you understood the explanation of the differences between the two main mxml files given above, the comments in Listing 5 and Listing 6

should serve as a sufficient explanation of the code in Listing 5 and Listing 6.

Run the program

I encourage you to run this program from the web. Then copy the code from Listing 4 through Listing 6. Use that code to create your own projects. Compile and run the projects. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Resources

I will publish a list containing links to Flex resources as a separate document. Search for Flex Resources in the Connexions search box.

Miscellaneous

This section contains a variety of miscellaneous materials.

Note: Housekeeping material

- Module name: XML - Namespaces - Flex 4
- Files:
    - Flex0086a\Connexions\FlexXhtml0086a.htm

-end-

The Default Application Container - Flex 3 and Flex 4
Learn about some of the differences between Flex 3 and Flex 4. Learn how to write a Flex application that controls the color, color gradient, and transparency of the background of the main window when a Flex application is played in Flash Player.

Note: Click AppBackground01 and AppBackground04 to run the Flex 3 and Flex 4 projects discussed in this lesson. *(Click the "Back" button in your browser to return to this page.)*

Table of Contents

Preface

General

This lesson is part of a series of tutorial lessons dedicated to programming with Adobe Flex.

What is Adobe Flex?

The previous lessons in this series have primarily dealt with XML issues and have nibbled around the edges of Flex. At this point, it is time to get really serious about Flex and start understanding what Flex, *(as an XML programming language)* , is all about.

According to one of the pages on the Adobe website,

> *"Flex is a highly productive, free open source framework for building and maintaining expressive web applications that deploy consistently on all major browsers, desktops, and operating systems. While Flex applications can be built using only the free open source framework, developers can use Adobe Flex Builder software to dramatically accelerate development."*

Note:
Subsequent to the original publication of this lesson, Adobe has released Flex version 4 *(in addition to Flex version 3)* and has replaced the Flex Builder 3 IDE with a new IDE named Flash Builder 4.
Flash Builder 4 supports both Flex 3 and Flex 4.
As of June 2010, Flex Builder 3 is no longer available, but Flash Builder 4 is available for downloading and is free for educational use .
I have updated this lesson to accommodate the availability of Flex 4 and Flash Builder 4 in addition to Flex 3.

Although Flash Builder 4 is very similar to Flex Builder 3 from a user perspective, there are numerous differences between Flex 3 and Flex 4. I will touch on some of these differences in this lesson and will explore them in more detail in future lessons.

Download links

Download links for the free Flex 3 and Flex 4 frameworks, as well as the Flash Builder 4 IDE are provided in Resources .

Powerful but more complicated

Flex 4 is more powerful, but also significantly more complicated than Flex 3. Therefore, it isn't clear to me that developers will want to make an immediate switch to Flex 4. Therefore, I will continue publishing material on Flex 3 in addition to Flex 4.

Is there a tipping point?

Anything that can be programmed in Flex can also be programmed directly in ActionScript, but the reverse is not true. It occurs to me that at some point, as Flex becomes more complicated, there may be a tipping point where developers will simply switch to ActionScript instead of expending the effort necessary to learn about new more-complicated aspects of Flex.

One of several ways to create applications for Flash Player *(version 10 or later)*

Flex 3 and Flex 4 *(combined with ActionScript 3)* simply provide one of several ways that developers can produce downloadable applications that will run in Adobe's Flash Player *(version 10 or later)* or Adobe Air *(see Resources)* .

Running Flash Player

Very briefly, Adobe Flash Player normally runs applications in a web browser while Adobe Air runs applications in a stand alone mode from the

desktop. However, if you have installed Flex Builder 3 or Flash Builder 4, it is usually possible to cause Flash Player to start and run in a stand alone mode by double clicking on an swf file. *(See Figure 1.)* If double clicking doesn't work, you can locate and run the file named FlashPlayer.exe. Once the Flash Player is running, you can open and run other swf files.

Why do I care about Flex and Flash Player?

One of the fastest growing segments of the game programming industry is the area of social games that typically run in Flash Player. I currently teach a course titled *"Game Development using C++"* and another course titled *"Game and Simulation Programming I : C#"* based on Microsoft's XNA Game Studio. I have a long-term interest in possibly also teaching a course in *"Game Development using Flex and ActionScript."*

Flex and ActionScript

However, my more immediate interest is related to two other courses that I teach. One of those courses is an introduction to XML. The other course is Object-Oriented Programming using ActionScript 3.

Flex is an XML-based language that is used to produce *mxml* files. *(In fact, Flex is simply a shorthand way of writing ActionScript program files. When you compile Flex MXML files, they are automatically converted into ActionScript files before compilation.)*

The resulting ActionScript files, along with other resources, are compiled into swf files that can be executed in Flash Player or Adobe Air.

Most of the lab projects in my XML course require the students to create Flex projects using both Flex 3 and Flex 4. Most of the lab projects in my more advanced ActionScript programming course require the students to use Flex as a launch pad for their ActionScript programs.

Flex SDK versus Flash Builder

Although it is possible to create Flex applications using nothing more than the free open source Flex SDKs *(see Resources)* and a text editor, that can

be tedious. Flash Builder 4 helps you to write mxml files containing layout and controls and makes the process somewhat more enjoyable. It will probably also help you to be more productive.

A complete Flex application

A complete Flex application consists of

- One or more mxml files that may or may not embed ActionScript 3 code.
- None, one or more ActionScript 3 files.
- Various resource files such as image files, sound files, etc.

As mentioned earlier, these are compiled into a Flash swf file, which can be executed in either a Flash player or *(in some cases)* Adobe Air.

Division of responsibility

Flex provides the layout and control structure for the application in an XML format while ActionScript provides the program logic. *(Of course, the whole thing can be written in ActionScript if you choose to do so.)*

ActionScript is an issue for my XML students

While earlier versions of ActionScript may have been simple and easy to learn, ActionScript 3 is a fully object-oriented programming language with a complexity level on par with C++.

Students in my XML class aren't required to have any programming knowledge, and aren't expected to have the prerequisite knowledge that would qualify them to learn ActionScript 3 while learning XML at the same time. Therefore, the early lessons in this series concentrate on the use of mxml for layout and control and forego any requirement to understand ActionScript 3. As I mentioned earlier, I teach object-oriented programming using ActionScript in a different course.

Viewing tip

I recommend that you open another copy of this document in a separate browser window and use the following links to easily find and view the figures and listings while you are reading about them.

Figures

Listings

Supplemental material

I recommend that you also study the other lessons in my extensive collection of online programming tutorials. You will find a consolidated index at www.DickBaldwin.com .

General background information

Download Flex

I have provided download links in Resources for both the free open source Flex SDKs and Flash Builder 4.

Flash Builder 4 includes the SDKs for both Flex 3 and Flex 4 along with an IDE that is useful for creating Flex applications. Both the raw SDK downloads and the Flash Builder 4 download includes a debug version of the Adobe Flash Player.

Install the Flex SDK

Installation of the free open source Flex SDKs is a little complicated *(but I will explain how to do it in a future lesson)* . I have provided a link in Resources for Installation and Release Notes.

Installation of Flash Builder 4 is straightforward, at least that is true for Windows. The download is an exe file. Just execute the downloaded file and follow the installation instructions.

Getting started with the free SDKs

If you are using the free stand-alone SDKs, I will provide instructions for getting started creating applications with them in a future lesson. Those versions include a command-line compiler. You will need to create your mxml files using a text editor and compile them using the command-line compiler.

Getting started with Flash Builder 4

If you are using Flash Builder 4, I have provided several links in Resources that will help you get started using the IDE. Perhaps the quickest way to get started is to view some of the videos at the link titled Flex in a Week video training .

Create a new project

Once you have Flash Builder 4 running, pull down the **File** menu and select **New/Flex Project** . Enter the name of your new project into the dialog box, make certain that the radio button for **Web application** is selected, specify the compiler that you want to use *(Flex 3 or Flex 4)* and click the **Finish** button. If you wish, you can specify a disk location for your project other than the default location.

The project tree

A **directory tree** for your new project will be created having at least the following folders:

- .settings
- bin-debug
- html-template
- libs
- src

This directory tree appears in the upper-left panel of the IDE. *(Note that the tree shows some other nodes such as default package and Flex x.x, but they aren't actually disk folders.)*

Flash Player output for a new Flex project

The **bin-debug** folder will contain several files including one with an swf extension and one with an **html extension** . As I mentioned earlier , if you double-click the file with the swf extension, it should open in Adobe Flash Player looking something like Figure 1.

> *(Note that the default background color for a new Flex 3 project is the gray color shown in Figure 1 while the default background color for a new Flex 4 project is white.)*

Flash Player output for a new Flex project.

Figure

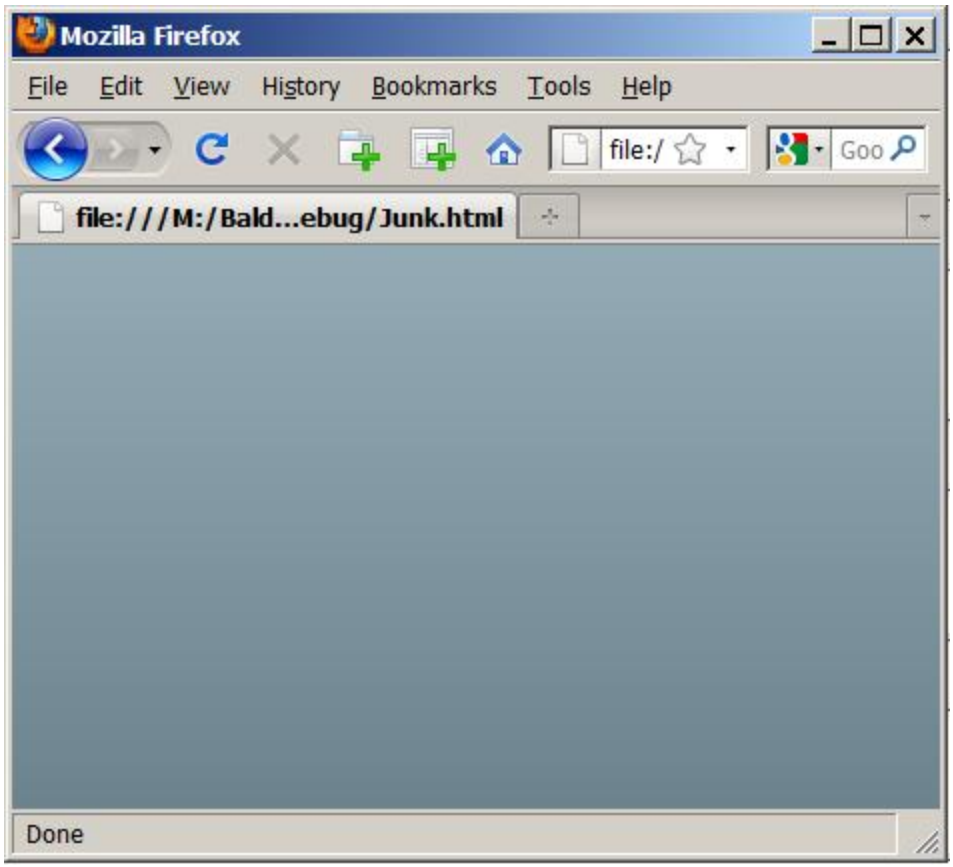
Flash Player output for a new Flex project.

Browser output for a new Flex project

When you installed Flash Builder 4, you should have been given an opportunity to update one or more browsers to include a special version of the Flash Player that supports debugging. If you open the html file mentioned above in one of those browsers, the output should look something like Figure 2 *(with either a gray or white background depending on which version of the Flex compiler was used to create the project)* .

Browser output for a new Flex project.

Figure



Browser output for a new Flex project.

The bin-release output

After you are satisfied with your program, you can create an output folder named **bin-release** by selecting **Project/Export Release Build...** The files in this folder are similar to the files in the **bin-debug** folder, but they may be smaller. This folder has been purged of debug information, and is the folder that is intended for deployment on a website.

Boring!

Granted, these outputs aren't all that exciting. However, they should help to confirm that you have Flash Builder 4 and Flash Player *(version 10 or later)* properly installed and linked to your browser.

The src folder

The folder in the directory tree mentioned earlier that will probably command most of your attention is the folder named src. When you create a new Flex project, this folder will contain a file with an extension of mxml. If you double click on that file name in the upper-left IDE window, the mxml file will open in the text editor window of the IDE. If you selected the Flex 3 compiler, the skeleton code will look something like Listing 1. *(It will be different for Flex 4, but I will get into that later.)*

A skeleton mxml file

Listing 1 shows the skeleton mxml code for a new Flex 3 project. This is the XML code that you will modify and supplement as you add features to your project.

Example:
Skeleton mxml code for a new Flex 3 project.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="absolute"
  minWidth="955"
  minHeight="600">

</mx:Application>
```

Compile and run the project

You can compile and run the new project by selecting either **Run** or **Debug** on the **Run** menu. *(The IDE provides a few other ways to run the project as well.)* The new project should open in your browser looking pretty much

like Figure 2. *(Remember, gray background for Flex 3 and white background for Flex 4.)* If you selected **Debug** , some debug text information should be displayed in the **Console** tab at the bottom of the IDE.

Preview

In this lesson, I will explain how to create two Flex projects. One is a simple Flex 3 project named **AppBackground01** . The other is a *(not so simple)* Flex 4 project named **AppBackground03** that replicates the output from the Flex 3 project.

Both projects are designed to produce the same output. To create the project using Flex 3, you will simply add attributes to the **Application** element shown in Listing 1, resulting in the XML code shown in Listing 2. However, you will need to do more than that to create the project in Flex 4. *(Remember, Flex 4 is more powerful but also more complicated to use.)*

Finally, I will show you the output from another Flex 4 project and challenge you as the student to write a Flex 4 project that produces a matching output.

More complex Flex projects

In general, to create more complex Flex projects, you will also need to add child elements, *(which may or may not include attributes)* , to the **Application** element.

To incorporate logic in your Flex projects, you will need to learn how to write some ActionScript code and embed ActionScript code in your Flex code *(or integrate ActionScript code in other ways)* .

Layout versus behavior

As I mentioned earlier, you can develop your Flex applications using the XML-based Flex language to establish the layout for your application.

Although it is possible to use Flex alone to provide simple behavior for your applications *(simple event handling for example)* , you will probably need to use ActionScript 3 to produce more complex behavior for your applications. *(Even the simple behavior requires some ActionScript, but it's not so obvious where the Flex code ends and the ActionScript code begins.)*

The layout capability in Flex 3 consists of containers *(that may contain other containers)* with names like **Canvas** , **HBox** , **VBox** , etc. The containers can also contain controls with names like **Button** , **CheckBox** , **ComboBox** , etc.

The text editor

The upper-middle pane in the Flash Builder 4 IDE has two tabs labeled **Source** and **Design** . If you select the **Source** tab, this upper-middle pane becomes a text editor in which you can edit XML and ActionScript code. In addition, the lower-left pane becomes an outline showing the hierarchical structure of your XML or ActionScript code.

The visual design editor

If you select the **Design** tab, the upper-middle pane becomes a visual design editor and the lower-left pane becomes a toolbox containing a variety of *components* including Controls, Layouts, Navigators, and Charts. *(Note that there are differences in the component sets for Flex 3 and Flex 4.)* In this mode, you can drag components from the lower-left pane into the design pane and make physical adjustments to their location, length, width, etc.

In addition, in design mode, the lower-right pane becomes a property window. You can select a component in the design pane and then set various properties such as font, color, etc., in the properties window.

XML code is automatically updated

When you drag a component into the design pane and set properties on that component, the XML code is automatically updated to reflect the addition of that component with the specified properties. Therefore, when using

Flash Builder 4 in design mode, you can largely avoid having to write raw XML code.

Flash Builder 4 versus the free open-source SDK

This visual design capability is probably the most important feature of Flash Builder 4 that separates it from the free open-source Flex 3 and Flex 4 SDKs. Flash Builder 4 lets you design your layout visually and it writes much of the XML code for you.

Documentation

The Adobe site provides extensive online documentation for Flex, Flash Builder, and ActionScript. You will find numerous links to documentation in Resources . This is where you go to learn how to use the Flex language in detail.

Documentation format

In its default form, much of the documentation is divided into three frames, two stacked vertically on the left and one on the right. If you select **All Packages** and **Frames** *(or something similar)* at the top of the rightmost frame, you will see Flex XML elements *(classes)* listed in the lower-left frame.

Select *No Frames*

If you select **No Frames** at the top, *(which is not always available)* the two frames on the left will disappear. This is useful if you want to use the **Find** capability of your browser to search for a word in the right frame. You can restore the three-frame format by selecting **Frames** at the top of the page.

Select a class

If you select a class, such as **Application** , in the lower-left frame, information about that class appears in the rightmost frame. If you find two or more classes with the same name in the lower-left frame, one probably refers to the Flex 3 version of the class and the other probably refers to the

Flex 4 version of the class. You must be very careful to make certain that you know which you are reading about. Reading about one and thinking that you are reading about the other can lead to very frustrating programming errors.

Styles

Many of the components in Flex 3 have a variety of styles that can be applied to the component either through the specification of attributes in the XML element or the use of a **Style** element that resembles a style sheet. *(That is much less true for Flex 4, which takes a different approach.)* The Flex 3 example that I will explain in the next section will illustrate the relationship among the following properties and styles for the **Application element** :

- backgroundColor
- backgroundGradientColors
- backgroundAlpha

Discussion and sample code

The **Application** element *(see Listing 2)* is a container into which you can place other components including other containers. Future lessons will be concerned with the containment properties of the **Application** element. In this lesson, I will concentrate on the properties and styles listed above that can be applied to that element.

Run the program

Before continuing, I suggest that you run the program named **AppBackground01** to familiarize yourself with the screen output.

The Flex 3 version

The mxml file

An XML listing for the Flex 3 application named **AppBackground01** is shown in Listing 2.

Example:
Flex 3 application namedAppBackground01.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
xmlns:mx="http://www.adobe.com/2006/mxml"

        backgroundColor="#FF0000"
        backgroundGradientColors="[#00FF00,
#0000FF]"
        backgroundAlpha="1.0">
</mx:Application>
```

Background color, color gradient, and transparency

The output for this application is similar to that shown in Figure 2 except that in this case, the color, color gradient, and transparency for the background is being controlled through the use of the following attributes:

- backgroundColor
- backgroundGradientColors
- backgroundAlpha

The code in Listing 2 uses the **backgroundColor** attribute of the **Application** element to set the background color of the Flash window to pure red using the hexadecimal notation #FF0000.

A few words about color

For those who may not be familiar with this concept, the overall background color is controlled by a mixture of different contributions of the

red, green, and blue primary colors. For example, pure red plus pure blue produces a color commonly known as *magenta* . Pure green plus pure blue produces *cyan* . Pure red plus pure green produces *yellow* . Pure red plus pure green plus pure blue produces *white* . The absence of all three colors produces black.

Three eight-bit bytes

The values for red, green, and blue respectively are specified by the values of three eight-bit bytes. The value of each byte can range from 0 to 255 decimal *(00 to FF in hexadecimal)* . A value of 0 *(00 in hexadecimal)* means that the primary color is not included in the mixture and a value of 255 *(FF in hexadecimal)* means that the primary color is included full strength in the mixture.

Using this scheme, it is possible to generate more than sixteen million different colors.

The order of the red, green, and blue contributions to the overall color in hexadecimal notation is RRGGBB, where the letter pairs stand for red, green, and blue respectively.

A color gradient

The Flex 3 application shown in Listing 2 sets the **backgroundGradientColors** attribute of the **Application** element to range from pure green *(#00FF00)* at the top of the Flash window to pure blue *(#0000FF)* at the bottom. As mentioned earlier, the application sets the **backgroundColor** to pure red using the hexadecimal notation #FF0000.

> *As you will see later, the **Application** element of a Flex 4 application doesn't have an attribute named **backgroundGradientColors** , so we will have to find another way to create a color gradient in the Flash window for a Flex 4 application.*

Alpha

The appearance of the resulting mix of colors can be controlled by changing the value of the **backgroundAlpha** attribute of the **Application** element. This attribute can take on values ranging between 0.0 and 1.0. *(In other programming systems, alpha is often allowed to vary between 0 and 255 decimal, or between 00 and FF hexadecimal.)*

Transparency or opacity

For those not familiar with the concepts surrounding screen color and transparency, the value of alpha specifies a transparency or opacity level. When an attempt is made to draw a pixel in the same screen location as an existing pixel, the color of the last pixel drawn will be completely opaque and will dominate if it has an alpha value of 1.0. In that case, the first pixel drawn will no longer be visible.

If the last pixel drawn has an alpha value of 0, it will be totally transparent and the first pixel drawn will continue to be totally visible.

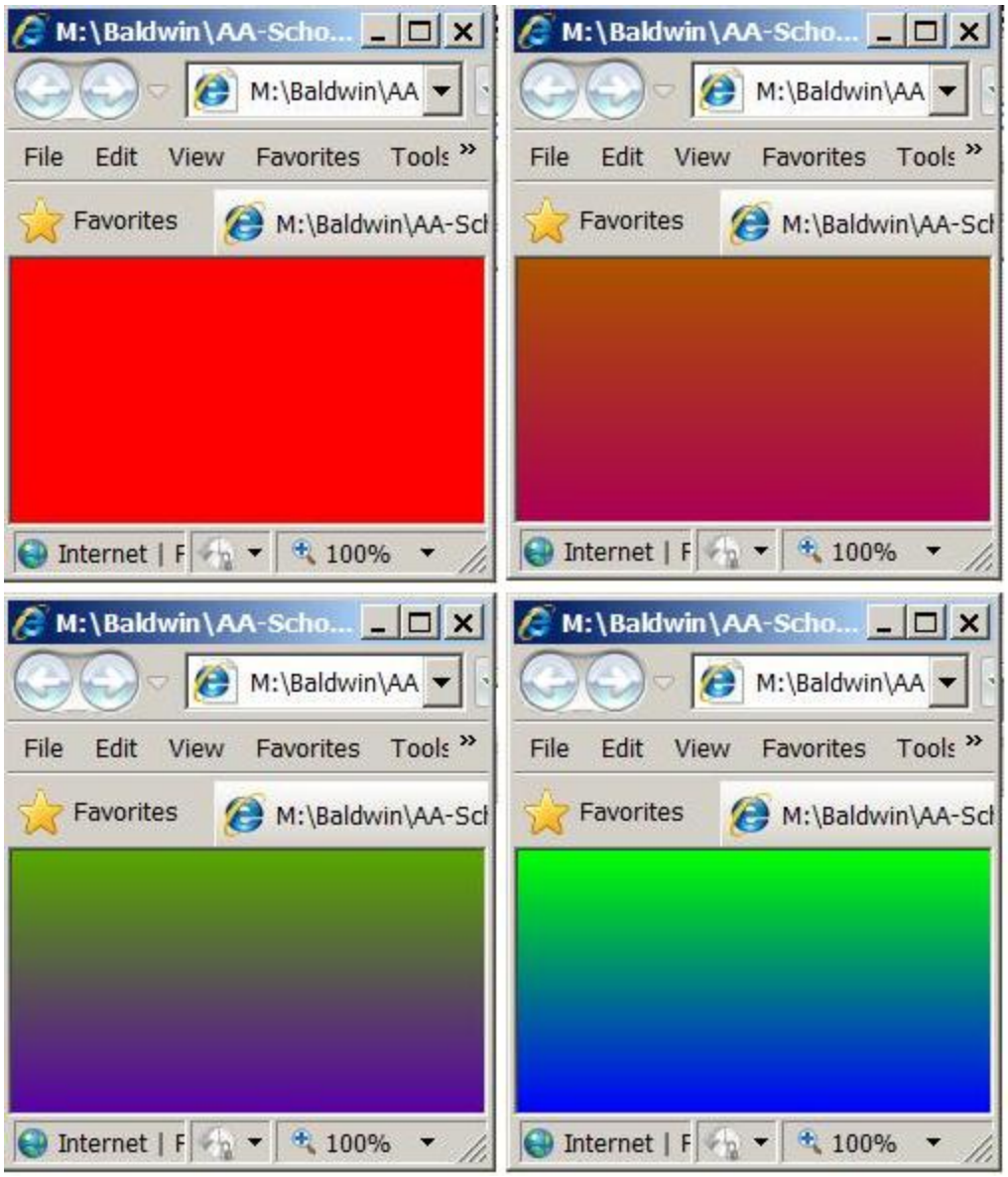
For values between 0.0 and 1.0, the first pixel drawn will show through to some extent and the final color of the pixel will be a mix of the original color and the new color.

Four runs with different alpha values

For this example, the Flex 3 application was modified, compiled, and run four times in succession by substituting four different values of **backgroundAlpha** into the code in Listing 2. The results are shown in Figure 3.

Application window background for alpha values of 0.0, 0.33, 0.66, and 1.0.

Figure



Application window background for alpha values of 0.0, 0.33, 0.66, and 1.0.

Four output images

The four images shown in Figure 3, going from left to right, top to bottom, represent alpha values of 0.0, 0.33, 0.66, and 1.0 respectively.

An alpha value of 0.0

For an alpha value of 0.0, the output color is pure red as shown by the top-left image in Figure 3. In this case, the gradient color scheme is totally transparent.

An alpha value of 1.0

For an alpha value of 1.0, the output colors range from pure green at the top to pure blue at the bottom as shown in the bottom-right image in Figure 3. In this case, the gradient color scheme is totally opaque and the red background color doesn't show through at all.

Alpha values of 0.33 and 0.66

For alpha values of 0.33 and 0.66, the output is a mix of the red background and the green/blue gradient as shown by the top-right and bottom-left images in Figure 3.

> *(The image in the bottom-left corner of Figure 3 corresponds to what you should see if you run the online version of this application.)*

Conclusion regarding the Flex 3 version

These results suggest that the alpha value specified by **backgroundAlpha** in Flex 3 applies to the colors specified by **backgroundGradientColors** and does not apply to the color specified by **backgroundColor** . .

In other words, the gradient colors can be made more or less transparent by changing the value of **backgroundAlpha** allowing the red **backgroundColor** to show through the gradient colors. However, the value of alpha doesn't appear to affect the background color.

If you modify the code in Listing 2 by removing the **backgroundColor** attribute, and then compile and run it for different values of alpha, you will see a similar result. In that case, however, the resulting colors will be a

mixture of the default gray background color shown in Figure 1 and the green/blue gradient shown in the bottom right corner of Figure 3.

The Flex 4 version

The Flex 4 program named **AppBackground03** replicates the behavior of the Flex 3 version named **AppBackground01** . By necessity, this version of the program is significantly more complicated than the Flex 3 version.

The skeleton code

The skeleton code for a Flex 4 project is shown in Listing 3.

Example:
Skeleton mxml code for a new Flex 4 project.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application
xmlns:fx="http://ns.adobe.com/mxml/2009"

xmlns:s="library://ns.adobe.com/flex/spark"

xmlns:mx="library://ns.adobe.com/flex/mx"
               minWidth="955"
               minHeight="600">
  <fx:Declarations>
    <!-- Place non-visual elements (e.g.,
services,
    value objects) here -->
  </fx:Declarations>

</s:Application>
```

This is not the first time that you have seen this code. I discussed the skeleton code for a Flex 4 project in an earlier lesson having to do with namespaces in Flex 4. Therefore, Listing 3 shouldn't require any further discussion

The mxml file for AppBackground03

An XML listing for the Flex 4 application named **AppBackground03** is shown in Listing 4.

Example:
Flex 4 application named AppBackground03.

```
<?xml version="1.0" encoding="utf-8"?>
<!--AppBackground03.mxml
This is a Flex 4 replica of the Flex 3 project
named
AppBackground01.
-->
<s:Application
    xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    backgroundColor="#ff0000">

  <s:Rect width="100%" height="100%">
    <s:fill>
      <s:LinearGradient rotation="90">
        <s:entries>
          <s:GradientEntry color="0x00FF00"
                           ratio="0.0"
                           alpha="0.66"/>
          <s:GradientEntry color="0x0000FF"
                           ratio="1.0"
                           alpha="0.66"/>
        </s:entries>
      </s:LinearGradient>
```

```
    </s:fill>
  </s:Rect>
</s:Application>
```

Essentially the same output

This Flex 4 code produces essentially the same visual output as the Flex 3 code shown in Listing 2. As you can see, however, this code is much more complicated than the code in Listing 2. *(Remember, Flex 4 is more powerful but is also more complicated.)* Not only is the code more complicated, but the concepts behind the code are also more complicated.

A style named backgroundGradientColors in Flex 3

The Flex 3 **Application** class has a property named **backgroundColor** , and a large number of styles including **backgroundGradientColors** and **backgroundAlpha** . The code in Listing 2 uses that property and those styles to control the appearance of the Flash window in Flex 3. Of particular significance here is the style named **backgroundGradientColors** .

No style named backgroundGradientColors in Flex 4

The Flex 4 **Application** class also has a property named **backgroundColor** . The value of that property is set to red in Listing 4. It also has a style named **backgroundAlpha** but I'm not sure what its purpose is. However, it does not have a style named **backgroundGradientColors** . Therefore, some other way must be found to implement a gradient color scheme in Flex 4 in order to replicate the behavior of the Flex 3 project shown in Listing 2.

Replicating the behavior of AppBackground01

There may be a variety of ways to accomplish this. One of those ways is to cover the entire Flash window with a rectangle and cause the rectangle to exhibit the desired gradient characteristics. This is illustrated in Listing 4.

Listing 4 begins by adding a **Rect** object whose dimensions cover one-hundred percent of the Flash window. *(Note that all of the elements in Listing 4 are in the spark namespace named "s".)*

What is a fill element?

Unfortunately, this is the point where you need become heavily involved in the concepts of object-oriented programming in ActionScript for the code in Listing 4 to make sense. I apologize in advance for the explanation that follows. However, I don't know any other way to explain it, so I will explain it using a combination ActionScript OOP terminology and mxml terminology.

What is a Rect element?

Rect is a class that represents a rectangle in the ActionScript package named **spark.primitives** . The mxml code in Listing 4 causes an object of the **Rect** class to be placed in the Flash window in front of the white default background. As mentioned above, the dimensions of the rectangle are such as to cover the entire Flash window, even when the size of the window is changed by the user. *(Note that the name **Rect** begins with an upper-case character. That is a clue as to what is going on.)*

Once again, what is a *fill* element?

Note that the name of this element begins with a lower-case character. That is also a clue as to what is going on. The **fill** element refers to a property of the **Rect** object. According to the documentation , the value of this property must be of type **IFill** .

What is type IFill ?

IFill is the name of an ActionScript *interface* . *(Note that IFill begins with an upper-case character. Once again, that is a clue.)* Declaring that the

value of the property named **fill** must be of type **IFill** means that the value must be a reference to an object of any class that satisfies the interface requirements defined by **IFill** .

According to the documentation , the following classes satisfy that requirement:

- BitmapFill
- LinearGradient
- RadialGradient
- SolidColor

Add a LinearGradient object

The next thing that appears in Listing 4 is an element named **LinearGradient** . *(Note that LinearGradient begins with an upper-case character.)* According to the documentation , **LinearGradient** is a class in the ActionScript package named **mx.graphics** . *(It isn't clear how it can be accessed via the spark namespace, but as you can see in Listing 4, that is obviously the case.)*

What have we accomplished so far?

Up to this point, the interpretation of the code in Listing 4 is that we have:

1. Created a **Rect** object, set its width and height attributes to completely cover the Flash window, and added it to the Flash Window.
2. Created a **LinearGradient** object, set its **rotation** attribute to 90, and assigned its reference to the **fill** property of the **Rect** object.

The element named entries

The next thing that we see in Listing 4 is an element named **entries** that is added to the **LinearGradient** element. Once again, note that this name begins with a lower-case character.

The documentation tells us that the **LinearGradient** class has a property named **entries** . The documentation further tells us that the value of this

property must be an array containing references to objects of the class named **GradientEntry** . These objects *"define the fill pattern for the gradient fill."*

Two GradientEntry elements

Referring back to Listing 4, we see that the two inner-most elements are two elements named **GradientEntry** .

We can now update the list of things that we have accomplished to read as follows:

1. Created a **Rect** object, set its width and height attributes to completely cover the Flash window, and added it to the Flash Window.
2. Created a **LinearGradient** object, set its **rotation** attribute to 90, and assigned its reference to the **fill** property of the **Rect** object.
3. Created two **GradientEntry** objects, set the values of three attributes on each object with values that I will explain later, placed those objects in a two-element array, and assigned that array to the **entries** property of the **LinearGradient** object created earlier.

The rotation attribute of the LinearGradient object

Before getting into the attributes of the **GradientEntry** objects, let's go back and take a look at the *rotation* attribute of the **LinearGradient** object.

According to the documentation :

> The LinearGradient class lets you specify the fill of a graphical element, where a gradient specifies a gradual color transition in the fill color. You add a series of GradientEntry objects to the LinearGradient object's entries Array to define the colors that make up the gradient fill.

By default, the color transition is from left to right. However, by specifying a rotation value in degrees for the **rotation** property, you can cause the

color transition to take place along an invisible line that is rotated by that amount relative to the horizontal.

Setting the rotation attribute to 90 degrees in Listing 4 causes the invisible line to be vertical and causes the color transition to take place from the top to the bottom of the Flash window as shown in Figure 3.

The color attributes of the GradientEntry elements

According to the documentation , **GradientEntry** is a class in the **mx.graphics** package. *(Note that the name begins with an upper-case character.)* A **GradientEntry** element can define several attributes, including **color** , **ratio** , and **alpha** . Let's begin by taking a look at the **color** attribute.

The color attributes of the GradientEntry elements

The array of **GradientEntry** objects assigned to the **entries** property of the **LinearGradient** object can contain a large number of objects. Each object has a **color** property, which is black by default.

The color transition produced by the **LinearGradient** object will begin with the color property of the first object in the array, transition through the colors specified by each successive object, and end up at the color specified by the last object in the array.

Only two colors for this case

The code in Listing 4 places only two **GradientEntry** objects in the array. The color property for the first one is pure green and the color property for the second one is pure blue. That means that the color will transition from pure green at the top of the Flash window to pure blue at the bottom of the Flash window as shown by the bottom right image in Figure 3.

The alpha attributes of the GradientEntry elements

The **alpha** attributes shouldn't require much of an explanation. They mean essentially the same thing that the **backgroundAlpha** attribute means in

Listing 2. The difference is that in Listing 2, the same alpha value is applied to both colors involved in the color transition while in Listing 4, each color involved in the transition has its own alpha value.

Replicate the behavior of the Flex 3 program

Setting the pair of alpha values in Listing 4 to 0.0, 0.33, 0.66, and 1.0 and then recompiling and re-running the program for each value causes the Flash window to take on the same four appearances shown from left to right, top to bottom in Figure 3.

The ratio attributes of the GradientEntry elements

The **ratio** attributes are a little more difficult to explain than the **color** attributes or the **alpha** attributes.

The documentation describes the **ratio** property in the following way:

> Where in the graphical element, as a percentage from 0.0 to 1.0, Flex samples the associated color at 100%. For example, a ratio of 0.33 means Flex begins the transition to that color 33% of the way through the graphical element.

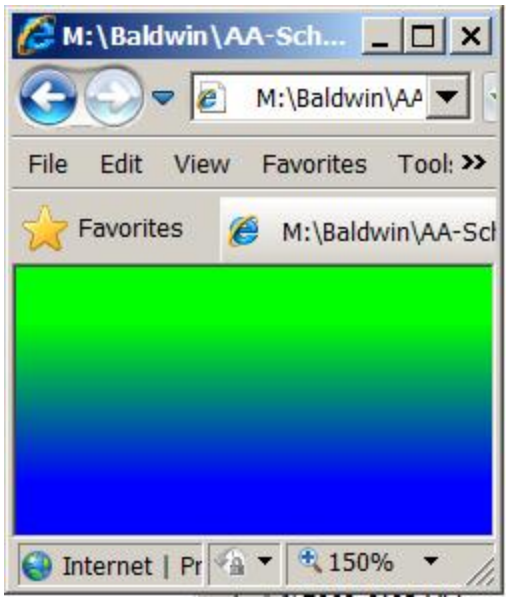
A picture is worth...

Let's see if I can illustrate this with a picture. I will change both **alpha** values in Listing 4 to 1.0 to cause the gradient colors to be totally opaque. Then I will change the **ratio** value to 0.2 for the first *(green)* **GradientEntry** object and change the **ratio** value to 0.8 for the second **GradientEntry** object.

The output produced by the program with these attribute values is shown in Figure 4.

Gradient output for narrow ratio zone.

Figure



Gradient output for narrow ratio zone.

Comparison with earlier results

The image shown in Figure 4 will be most meaningful if you compare it with the lower-right image in Figure 3. Although that image was actually produced by the Flex 3 program, the code in Listing 4 would produce the same output for **alpha** values of 1.0 and **ratio** values of 0.0 and 1.0 respectively.

For that case, there is a smooth gradient from the very top to the very bottom of the Flash window. In Figure 4, however, there is no gradient in the top twenty-percent or the bottom twenty-percent of the Flash window. Instead the entire gradient is squeezed in between those two limits.

End of this program

That ends my explanation of the Flex 4 project named **AppBackground03** shown in Listing 4. I hope that this explanation hasn't been too steeped in technical ActionScript details to be understandable, but as I told you earlier, I don't know any other way to explain it. In fact, if it were not for the fact

that I am very experienced in ActionScript object-oriented programming, I doubt that I could have understood, much less explained the code in Listing 4.

Upper case versus lower case

Oops, I almost forgot to explain the upper-case versus lower-case thing that I mentioned several times above.

Element names and class names

Objects in Flex mxml are represented by element names. Objects are created from classes and the mxml element names mirror the class names from which the objects are created. By convention, class names *(and interface names)* begin with an upper-case character in ActionScript.

Properties and styles

Most objects have properties *(such as the **color** property of a **GradientEntry** object)* and some objects have styles *(such as the **backgroundColor** style of an **Application** object)* . By convention, property and style names in ActionScript begin with a lower-case character.

Representation of properties in Flex mxml code

Typically, object properties and styles are represented by attributes having the same names in Flex mxml. However, as illustrated by Listing 4, in some cases, properties can also be represented by elements in Flex 4. *(I don't recall having seen this in Flex 3 but that doesn't mean that it is not possible.)*

Conclusions regarding upper-case versus lower-case

If you see an element name that begins with an upper-case character, *(such as **Rect** , **LinearGradient** , and **GradientEntry** in Listing 4)* , that probably means that it represents an object.

If you see an element name that begins with a lower-case character, *(such as **fill** and **entries** in Listing 4)* , that probably means that it represents a

property.

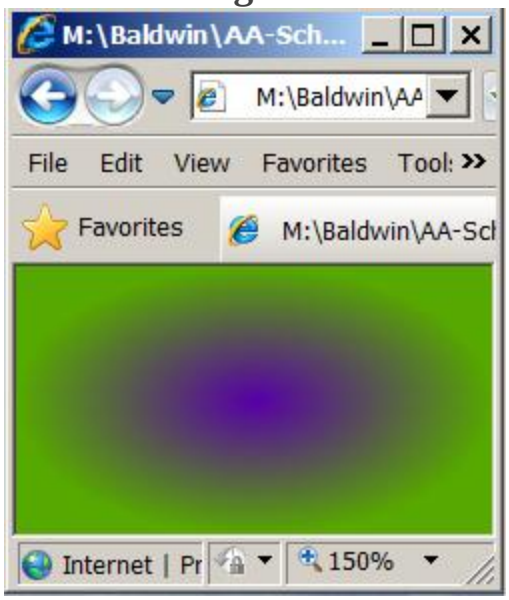
Another Flex 4 application

I told you earlier that in addition to the LinearGradient class, another of the classes that satisfy the interface requirements of the interface named **IFill** is RadialGradient .

Figure 5 shows the Flash window for a Flex 4 project that I wrote using the **RadialGradient** class.

Flash window for Flex 4 project named AppBackground04.

Figure



Flash window for Flex 4 project named AppBackground04.

An exercise for the student

However, this time I am not going to explain the code that produced the output shown in Figure 5. Instead, I am going to leave it as an exercise for

the student to dig into the documentation and figure out how to write the code to produce that output on her own.

Run the programs

I encourage you to run the online versions of the programs that I have explained in this lesson. Then copy the code from Listing 2 and Listing 4. Use that code to create Flex projects of your own. Compile and run your projects. Experiment with the code, making changes, and observing the results of your changes. For example, you might try changing the value of the rotation attribute in Listing 4 to see what that does. Make certain that you can explain why your changes behave as they do.

Resources

I will publish a list containing links to Flex resources as a separate document. Search for Flex Resources in the Connexions search box.

Miscellaneous

Note: Housekeeping material

- Module name: The Default Application Container - Flex 3 and Flex 4
- Files:
    - Flex0102\Flex0102.htm
    - Flex0102\Connexions\FlexXhtml0102.htm

you should be aware that some of the HTML elements in this module may not translate well into PDF.

-end-

Using Flex 3 in a Flex 4 World
Learn how to continue using Flex 3 even after the release of Flash Builder 4 and the disappearance of Flex Builder 3.

Note: Click test to run this Flex program. *(Click the "Back" button in your browser to return to this page.)*

Table of Contents

Preface

General

This lesson is part of a series of tutorial lessons dedicated to programming with Adobe Flex. The main purpose of this lesson is to supplement the earlier lesson titled The Default Application Container - Flex 3 and Flex 4 with more information about the **Flash Builder 4** IDE as well as information about the **FlashDevelop** IDE.

Adobe released a new product named **Flash Builder 4** in March of 2010 and immediately removed all, or at least most of the references to **Flex Builder 3** from their website. Those references that were not removed were converted to references to Flash Builder 4. For example, the link *(see Resources)* that previously opened an Adobe page for free downloading of Flex Builder 3 *(for educational use)* now opens a page for free downloading of Flash Builder 4 for educational use.

What is Flash Builder 4?

Flash Builder 4 is an apparently upgraded version of Flex Builder 3 with a new name. Flex Builder 3 was based on Flex version 3, while Flash Builder 4 is primarily based on Flex version 4. However, Flash Builder 4 still supports Flex 3 in addition to Flex 4.

Regardless of how you choose to get there, the objective is to create swf files that can be executed in the Flash Player. Flash Builder 4 is only one of several different ways to create applications that will run in the Flash Player.

The Flex 4 SDK and Flash Player 10

In conjunction with the release of Flash Builder 4, Adobe also released a new version *(version 4)* of the free open-source Flex SDK and a new version *(version 10)* of the Flash Player.

Flex Builder 3 was based on Flex 3 and required Flash Player 9 *(or later)* for execution of the swf files produced by compiling a Flex application.

Flash Builder 4 is primarily based on Flex 4 and requires Flash Player 10 *(or later)* for execution of the resulting swf files.

Flash Builder 4 supports Flex 3

Fortunately, Flash Builder 4 still supports Flex 3 for backward compatibility. I will show you later in this tutorial how to use Flash Builder 4 with Flex 3.

Should you use Flex 4?

If you are using Flex to create web applications in a professional capacity, you should probably become familiar with Flex 4. It appears to provide some features that are not available in Flex 3.

More powerful but also more complicated

Flex 4 is more powerful than Flex 3. However, Flex 4 is also more complicated than Flex 3. In some cases, Flex 4 also appears to create much larger swf files than Flex 3 for solutions to the same problems. Therefore, I suspect that some developers will not make an immediate switch to Flex 4, and that Flex 3 will continue to be commonly used for a few more years.

Beginning in the Fall 2010 semester, I will upgrade my Introduction to XML course to include both Flex 3 and Flex 4.

Flex is a cryptic shorthand programming language

The important thing to remember is that Flex *(versions 3 and 4)* is simply cryptic xml-based shorthand programming languages that can be used to create some aspects ActionScript programs. You should also remember that anything that can be programmed in Flex can be programmed directly in ActionScript. However, the reverse is not true.

When Adobe released Flex 4 and Flash Builder 4, they actually added a large number of classes to the ActionScript application programming interface *(API)* . So, the real question is not whether you should use Flex 4. The real question is whether the new classes that were added to the API are useful in your application. If so, use them. If not, stick with the classes that were part of the API before the new classes were added.

Understanding Flex

If you really want to understand Flex *(instead of simply using it in cookbook fashion)* you need to learn about object-oriented programming using ActionScript.

Viewing tip

I recommend that you open another copy of this document in a separate browser window and use the following links to easily find and view the figures and listings while you are reading about them.

Figures

Listings

Supplemental material

I recommend that you also study the other lessons in my extensive collection of online programming tutorials. You will find a consolidated index at www.DickBaldwin.com .

Preview

You learned how to create a Flex project named **AppBackground01** using Flex Builder 3 in the earlier lesson titled The Default Application Container - Flex 3 and Flex 4 . However, Flex Builder 3 is no longer available from the Adobe website.

Therefore, in this lesson, I will explain how to create the same Flex project in three different ways. However, in this lesson, I will shorten the name of the project by changing the project name to **test** .

Discussion and sample code

Run the program

Before continuing, I suggest that you run the program to familiarize yourself with the screen output.

Different ways to create Flex 3 applications

Here is a list of four different ways that you can create Flex 3 applications:

1. Use **Flex Builder 3** if you already have it or can find it on the web.
2. Download and use the free open-source **Adobe Flex 3 SDK** *(see Resources)*.
3. Download and use the free **FlashDevelop IDE** *(see Resources)*
4. Download and use **Flash Player 4** with the **Flex 3 SDK** *(see Resources)*

Download and use the free open-source Adobe Flex 3 SDK

It is relatively easy to use the free open-source Flex 3 SDK to compile an mxml document and create a swf file that can be run stand-alone in the Flash Player. It is more difficult to deploy that swf file on a server.

Compiling an mxml file from the command line

The main claim to fame for XML is that XML files are plain text files. An mxml file is an XML file. Therefore, you can create an mxml file containing mxml code, *(such as that shown in Listing 1)* , using any plain text editor.

(The code shown in Listing 1 is the same as the code used in the Flex application in the earlier lesson titled The Default Application Container - Flex 3 and Flex 4 .)

Example:
Flex application named test.mxml.

```
<?xml version="1.0" encoding="utf-8"?>

<mx:Application
xmlns:mx="http://www.adobe.com/2006/mxml"

  backgroundColor="#FF0000"
  backgroundGradientColors="[#00FF00, #0000FF]"
  backgroundAlpha="1.0">

</mx:Application>
```

A test program

Assume that you have created such a file named **test.mxml** and that you have stored the file in the following folder on a Windows system:

```
C:\jnk\1\test.mxml
```

Download the zip file containing the Flex 3 SDK *(see Resources)* and extract the contents of the zip file into a folder of your choosing. That folder will then contain the following file:

```
...\bin\mxmlc.exe
```

Compile the test program

If you open a command-line window in the **bin** folder and execute the following command on the command line, the mxml file will be compiled and a file named **test.swf** will be created in the same folder as the mxml file:

```
mxmlc c:\jnk\1\test.mxml
```

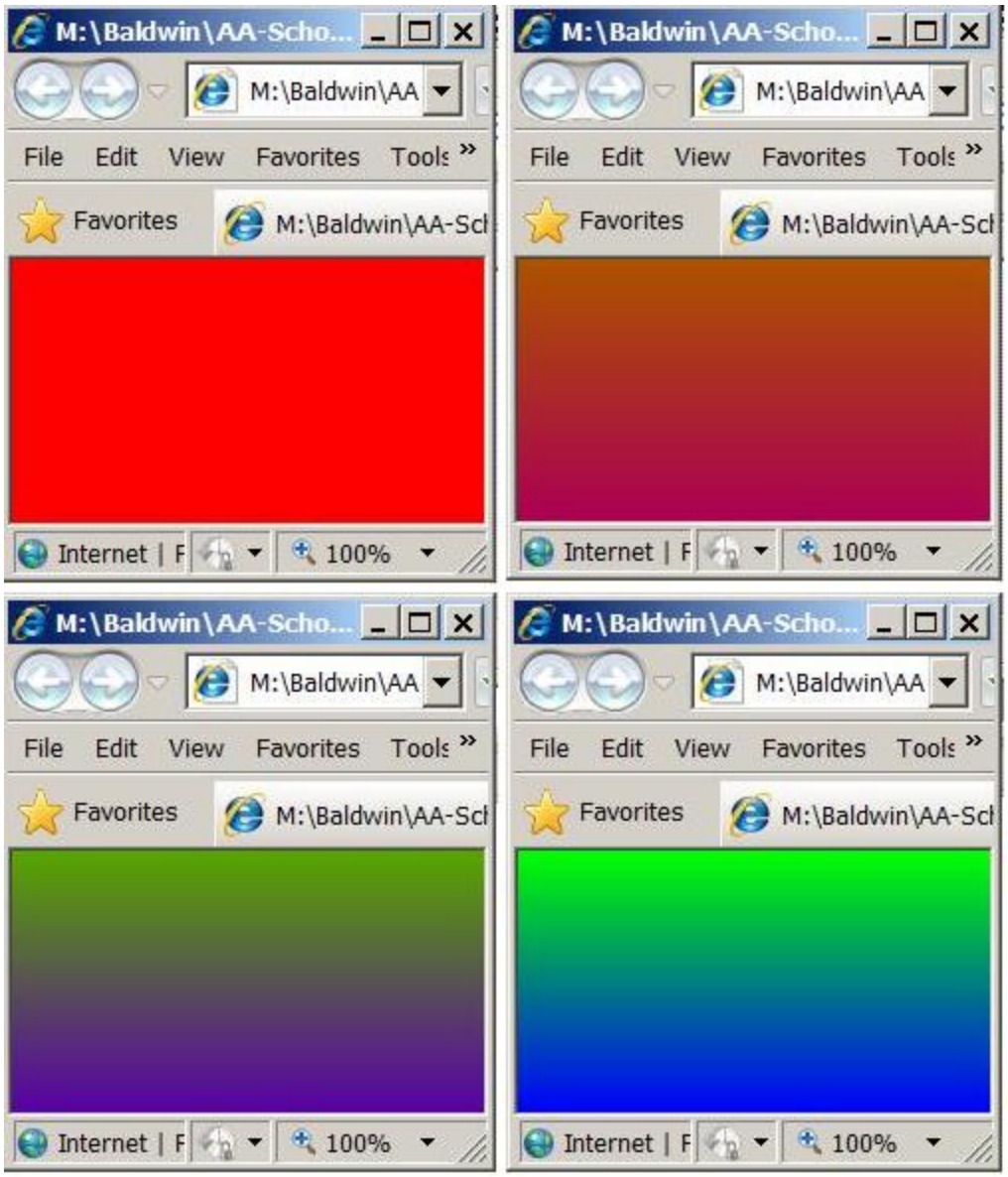
If you have a stand-alone Flash Player *(version 9 or later)* on your system, you can open the file named **test.swf** in that player.

Testing your installation

You should have no difficulty using this approach to modify the **backgroundAlpha** values in the mxml code shown in Listing 1 to produce swf files that produce the four gradient images shown in Figure 1 when run in a stand-alone Flash Player version 9 or later. *(Note that the four images shown in Figure 1 are actually screen shots of the swf files running in a browser.)*

Test program output for alpha values of 0.0, 0.33, 0.66, and 1.0.

Figure



Test program output for alpha values of 0.0, 0.33, 0.66, and 1.0.

Four runs with different alpha values

For this example, the mxml code shown in Listing 1 was modified, compiled, and run four times in succession by substituting four different values of **backgroundAlpha** into the code.

Four output images

The four images shown in Figure 1, going from left to right, top to bottom, represent alpha values of 0.0, 0.33, 0.66, and 1.0 respectively.

An alpha value of 0.0

For an alpha value of 0.0, the output color is pure red as shown by the top-left image in Figure 1. In this case, the gradient color scheme is totally transparent.

An alpha value of 1.0

For an alpha value of 1.0, the output colors range from pure green at the top to pure blue at the bottom as shown in the bottom-right image in Figure 1. In this case, the gradient color scheme is totally opaque and the red background color doesn't show through at all.

> *(The image in the bottom-right corner of Figure 1 corresponds to what you should see if you run the online version of this application as described earlier .)*

Alpha values of 0.33 and 0.66

For alpha values of 0.33 and 0.66, the output is a blend of the red background and the green/blue gradient as shown by the top-right and bottom-left images in Figure 1.

Deploying the swf file

If you don't have a stand-alone Flash Player on your system, or if you need to make the swf file accessible from a web site, you will need to *deploy* the swf file in order to cause it to run in a Flash Player browser plug-in .

The simplest case

In the simplest case, this will require you to create a JavaScript file and an html file. Templates for those two files are provided in the Flex SDK folder with the following names and locations:

```
...\flex_sdk_3.5\templates\no-player-
detection\AC_OETags.js

...\flex_sdk_3.5\templates\no-player-
detection\index.template.html
```

I will leave it as an exercise for the student to read the online deployment instructions *(see Application Deployment in Resources)* and learn how to deploy the swf file.

Download and use the free FlashDevelop IDE

If you prefer not to work at the command-line level, another option is to download and use the free **FlashDevelop** IDE *(see Resources)* .

Works with the Flex SDK

The FlashDevelop program works in conjunction with the free open-source Flex SDK to make it easier to create, compile, and deploy Flex applications than is the case when working from the command line. *(The version of the FlashDevelop program that is available in June 2010 is not compatible with the Flex 4 SDK.)*

See the Getting Started website

After installing the FlashDevelop program, you should open the **Getting Started** website *(see Resources)* where you will find instructions for configuring the program and getting it ready for use.

The FlashDevelop IDE at startup

When you first start the FlashDevelop program running, you will see an integrated development environment *(IDE)* that looks something like Figure

2.
The FlashDevelop IDE at startup.

Figure

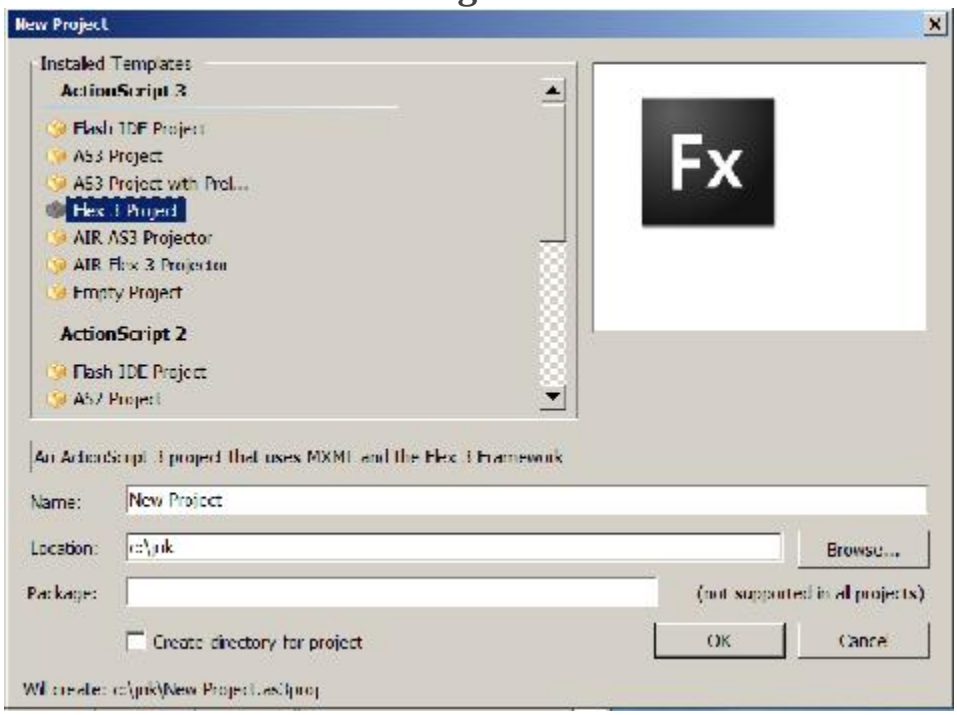
The FlashDevelop IDE at startup.

Create a new project

Selecting the *Create a new project* link on the right side of Figure 2 will open a dialog that looks much like the one shown *(in reduced form)* in Figure 3.

The new project dialog.

Figure



The new project dialog.

The text is difficult to read

Because of the reduced size, the text in Figure 3 is difficult to read. The highlighted item in the list in the upper left reads "Flex 3 Project."

Select Flex 3 Project

Then enter the name and location of the new project in the text fields near the bottom of Figure 3, and click the **OK** button. The IDE will change to look something like Figure 4.

The FlashDevelop IDE for a new project named test.

Figure

The FlashDevelop IDE for a new project named test.

The project tree

Note in particular the tree structure shown on the right side of Figure 4 that is rooted in **test (AS3)** . This tree structure replicates the tree structure that is created in the folder that was earlier specified as the location for the project.

Re-opening the project later

However, that folder also contains a file named **test.as3proj** that is not shown in Figure 4. You can double-click this file later to re-open the project in FlashDevelop.

Double-click the mxml file

The next step is to double-click the file shown in Figure 4 named **Main.mxml** . This will open that file in a text editor as shown in Figure 5.
FlashDevelop text editor showing skeleton code for a Flex 3 project.
Figure

FlashDevelop text editor showing skeleton code for a Flex 3 project.

Skeleton code

At this point, the text editor shown in Figure 5 contains the skeleton code for a Flex 3 project similar to what you saw in Listing 1 of the earlier lesson titled **The Default Application Container - Flex 3 and Flex 4** .

Modify and compile the program

Replace the skeleton code shown in Figure 5 with the code shown in Listing 1 above.

Then select **Build Project** on the **Project** menu.

Note the panel at the bottom of the IDE

You can drag and open a panel at the bottom of the IDE, if necessary, to expose the compiler output text.

Quite a lot of text will appear in that panel.

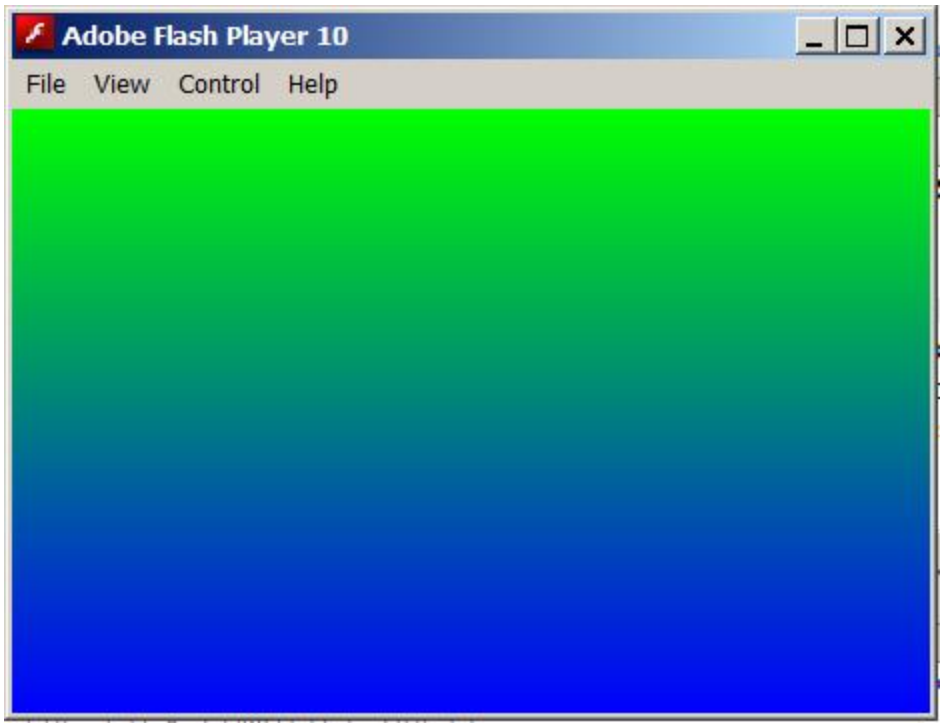
If everything goes well, the last line of text in that panel will end with the words *"Build succeeded"* .

Test the program

Select **Test Movie** on the **Project** menu. If all goes well, The Flash Player should open and display something very similar to Figure 6.

Test program output displayed in the Flash Player.

Figure



Test program output displayed in the Flash Player.

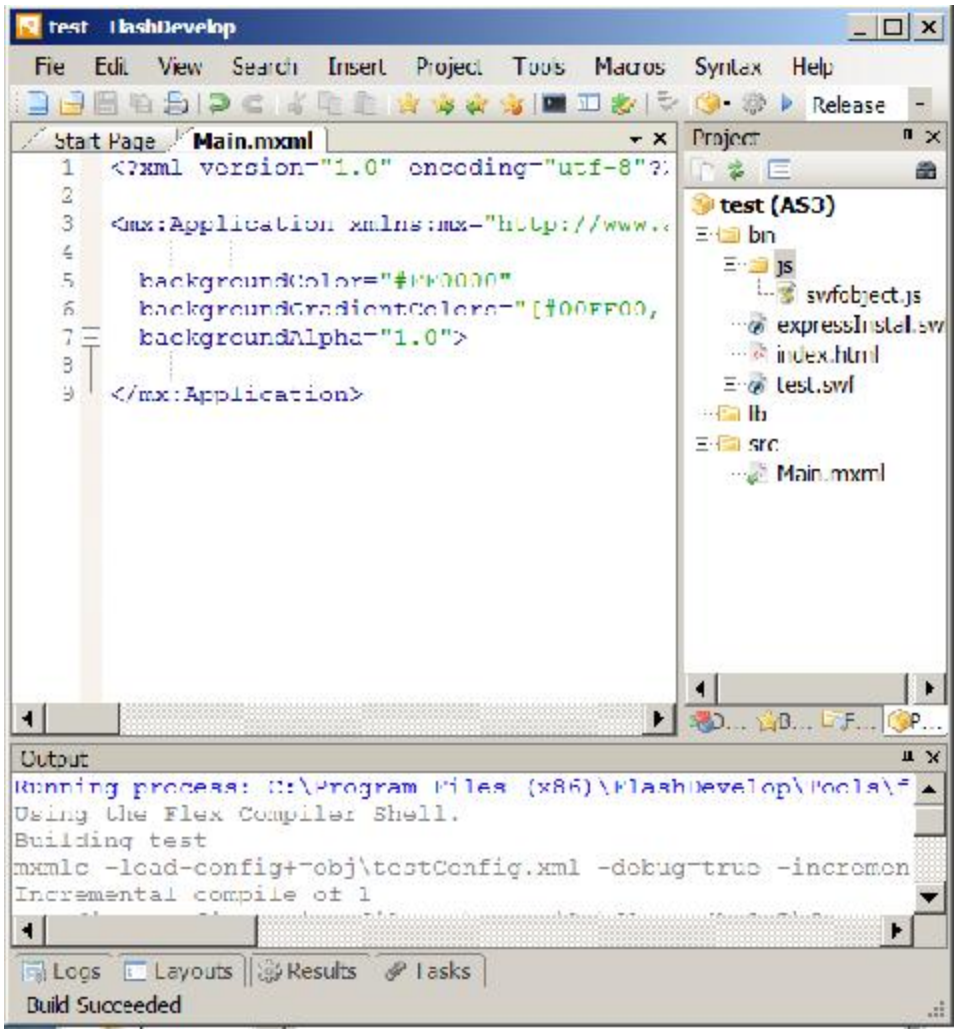
Create a Release Build

If you like what you see at this point, select **Release** in the pull-down list shown near the upper-right corner of Figure 7 and build the project again.

This will modify the contents of the folder named **bin** shown in Figure 4. This new version of the folder named **bin** can be deployed on a server.

FlashDevelop showing the Debug/Release pull-down list.

Figure



FlashDevelop showing the Debug/Release pull-down list.

Executing the Flex project

The folder named **bin** contains a file named **index.html** as well as a swf file. You can open that html file in a browser to play the Flash movie provided that the Flash Player plug-in, version 9 or later, has been installed in the browser.

The swf file can also be opened directly in a stand-alone Flash player, version 9 or later.

What is the difference between the Debug and Release builds?

While it is also possible to get an output that can be run in the Flash Player in **Debug** mode *(the opposite of **Release** mode)* , the output produced in **Release** mode is normally smaller and more compact.

Pros and cons of the FlashDevelop program

By now, you have probably concluded that the **FlashDevelop** program makes it easier to develop **and deploy** Flex programs than is the case for the developing such programs from the command line.

Deployment made simple

In addition to providing a nice text editor, the program also handles the tedious task of creating the files necessary to deploy the application and creates a folder named **bin** containing all of the necessary files.

The downside

The primary downside of FlashDevelop *(relative to Flex Builder 3 or Flash Builder 4)* , is that the FlashDevelop program does not provide the drag-and-drop visual design mode for GUI components that is provided by the two Adobe builder programs.

May or may not be important

Whether or not that is important will depend on the intended use of the IDE.

For example, students in my *Introduction to XML* course may consider that to be a serious shortcoming because most of the lab projects in that course involve layouts for GUI components.

On the other hand, students in my *ActionScript OOP* programming course shouldn't have much need for the drag-and-drop feature of the builder programs. That course concentrates more on programming theory and less on drag-and-drop GUI construction.

Download and use Flash Player 4 with the Flex 3 SDK

Since Flex Builder 3 is no longer available for download from the Adobe website, those students who need the drag-and-drop feature have little choice other than to use Flash Builder 4 in conjunction with Flex 3. Therefore, I will tell you what I have learned through experimentation with Flash Builder 4.

Two different versions

To begin with, Flash Builder 4 can be downloaded in at least two versions. One version is a stand-alone program. The other version is a plug-in for Eclipse. I am running the plug-in version, so some of the screen shots that follow may be different from what you would see with the stand-alone version.

Similar to Flex Builder 3

Many aspects of Flash Builder 4 are very similar to Flex Builder 3, which you learned about in earlier lessons. Therefore, I will concentrate on the differences that I have identified.

Switch to the Flash perspective

When you start Flash Builder 4 for the first time, you may be given an opportunity to *"Switch to the Flash Perspective."* If so, you should make the switch. Otherwise, things probably won't go well at all. *(Once I learned that I needed to make the switch, I never saw that option again.)*

Create a new Flex project

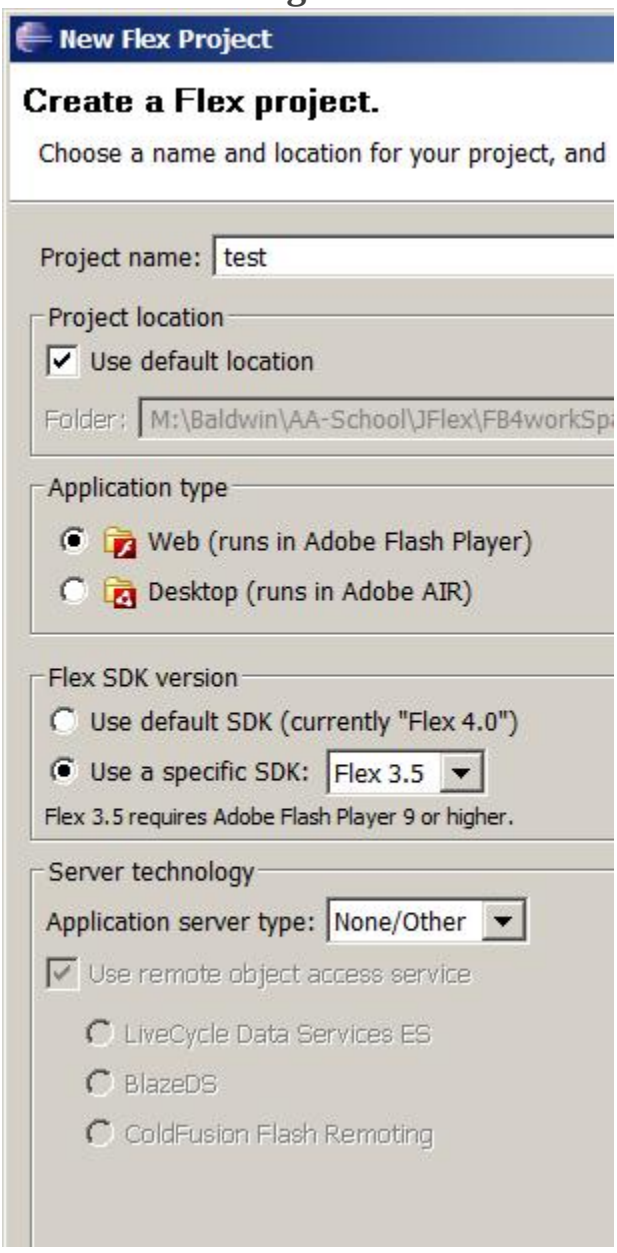
To create a new Flex project, pull down the **File** menu and select **New/Flex Project** .

Specify Flex 3.x

When the **New Flex Project** dialog appears, be sure to select **Web** and specify **Flex 3.x** *(Flex 3.5 as of June 2010 but this may change over time)* as shown in Figure 8.

The New Flex Project dialog in Flash Builder 4.

Figure



The New Flex Project dialog in Flash Builder 4.

The Application server type

Unless you know a good reason to do otherwise, you should accept the default server type of **None/Other** as shown in Figure 8.

Click the Finish button

After you provide the required information in the **New Flex Project dialog** , click the **Finish** button at the bottom of the dialog. *(Because of the way that I cropped the image, the Finish button is not shown in Figure 8.)*

Open the text editor

If necessary, select the **Source** tab at the top of the main panel to open the text editor in the main panel as shown in Figure 9. This is essentially the same as Flex Builder 3. *(The alternative is the **Design** tab, which I discuss below.)* Enter your mxml code in the text editor.

The Design tab

If you need to do drag-and-drop component layout, select the **Design** tab at the top of the main panel. This will cause the lower-left panel to become a component panel and will cause the lower-right panel to become a property panel in a manner similar to Flex Builder 3.

Build your project

Once you are satisfied with your component layout and your mxml source code, select **Build Project** on the **Project** menu.

Run your project

Then select **Run** on the **Run** menu. This may cause a dialog titled **Run As** to appear on the screen. *(I don't recall seeing this dialog in Flex Builder 3.)* If the dialog does appear, select **Web Application** and click the **OK** button.

This should cause the application to open and run in your default browser provided that the browser has the Flash Player plug-in, version 9 or later, installed.

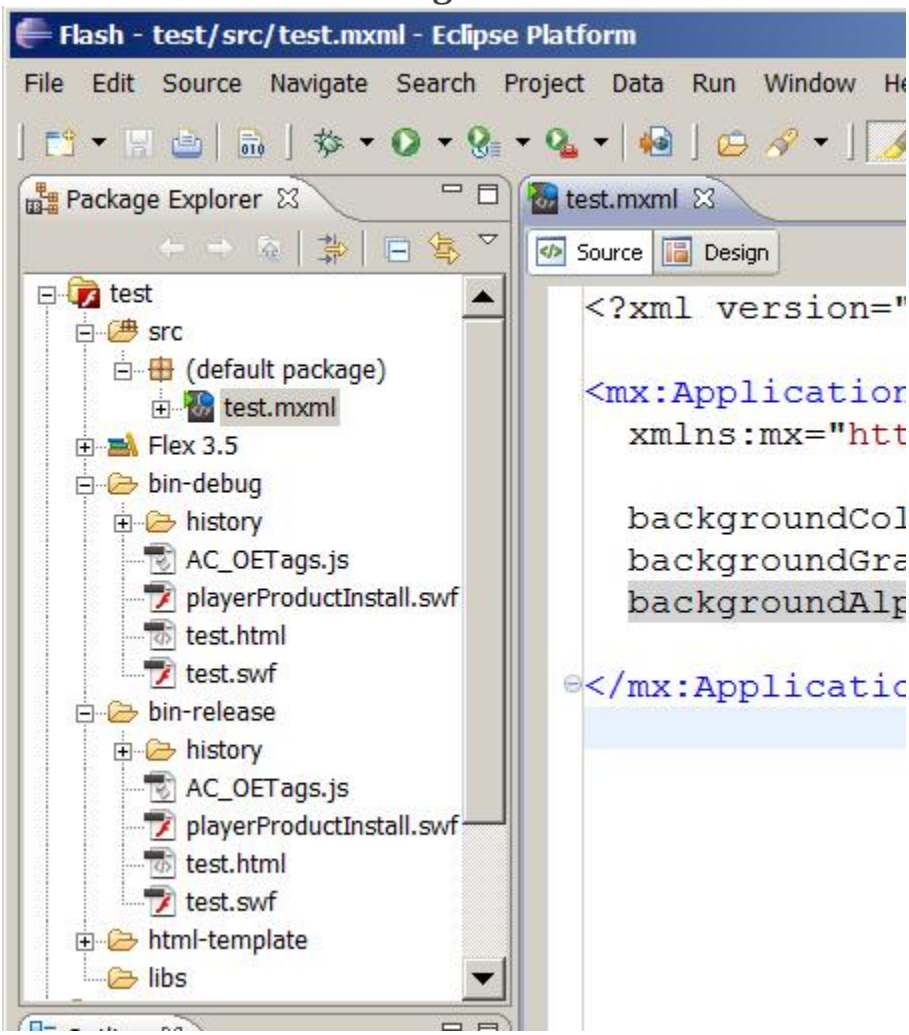
Create a Release Build

If you are happy with what you see at this point, select **Export Release Build...** on the **Project** menu. This will open another dialog titled **Export Release Build** which will probably contain the correct information. Click the **Finish** button on this dialog to cause the release build to be exported.

The bin-release folder

Clicking the **Finish** button causes the folder named **bin-release** *(shown in the project tree in Figure 9)* to be created and populated.
Contents of the project tree for the test program with Flash Builder 4.

Figure



Contents of the project tree for the test program with Flash Builder 4.

Deploying the project

The **bin-release** folder contains everything necessary to deploy the project on a server. Opening the file named **test.html** in that folder will cause the Flash program to be played in a Flash Player browser plug-in. In addition, the file named **test.swf** can be played in a stand-alone Flash player.

The bin-debug folder

The **bin-debug** folder that you see in Figure 9 also contains files named **test.html** and **test.swf** . They can be opened in a browser and in a stand-alone Flash player to produce essentially the same results.

A smaller movie file

However, the **test.swf** file in the **bin-release** folder will often be much smaller and more compact than the **test.swf** file in the **bin-debug** folder, resulting in shorter download times from the server.

Run the program

I encourage you to run the online version of the program. Then copy the code from Listing 1. Use that code to create Flex projects for all three approaches. Compile and run the projects. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Resources

I will publish a list containing links to Flex resources as a separate document. Search for Flex Resources in the Connexions search box

Miscellaneous

This section contains a variety of miscellaneous materials.

Note: Housekeeping material

- Module name: Using Flex 3 in a Flex 4 World
- Files:
    - Flex0103\Flex0103.htm
    - Flex0103\Connexions\FlexXhtml0103.htm

-end-

Handling Slider Change Events in Flex 3 and Flex 4
Learn how to write inline event handler code to handle slider change events.

Note: Click SliderChangeEvent01 , SliderChangeEvent02 , and SliderChangeEvent03 to run the Flex programs from this lesson. *(Click the "Back" button in your browser to return to this page.)*

Table of Contents

Preface

General

This lesson is part of a series of tutorial lessons dedicated to programming with Adobe Flex. The material in this lesson applies to both Flex 3 and Flex 4.

Viewing tip

I recommend that you open another copy of this document in a separate browser window and use the following links to easily find and view the figures and listings while you are reading about them.

Figures

Listings

- Listing 7 . Complete listing of SliderChangeEvent02.
- Listing 8 . Complete listing of SliderChangeEvent03.

Supplemental material

I also recommend that you study the other lessons in my extensive collection of online programming tutorials. You will find a consolidated index at www.DickBaldwin.com .

General background information

If you learn how to program using ActionScript 3, you will probably integrate large amounts of ActionScript code into your Flex projects to provide complex event handling. In the meantime, it is possible to provide simple event handlers in Flex by embedding a very small amount of ActionScript code in your Flex code. I will illustrate and explain that capability in this lesson.

Preview

I encourage you to run the online version of the programs from this lesson before continuing.

Three Flex projects

I will present and explain three Flex projects in this lesson. The first project is named **SliderChangeEvent01** . This project was first developed using Flex Builder 3 and later developed using the Flex 3 compiler and Flash Builder 4. The results were essentially the same in both cases. This project uses classes from the Flex 3 *(mx)* library exclusively. *(The screen shots shown in Figure 4, Figure 5, and Figure 6 are from Flex Builder 3.)*

The second project

The second project is named **SliderChangeEvent02** . This project was developed using the Flex 4 compiler and Flash Builder 4. This is a hybrid project that uses classes from both the Flex 3 *(mx)* library and the Flex 4 *(spark)* library. The behavior of this project is similar to the behavior of the other project, but they look different in several ways that I will explain later.

The third project

The third project is named **SliderChangeEvent03** . This project is a modification of the hybrid project in which the classes used are drawn exclusively from the Flex 4 spark library.

Order of upcoming explanations

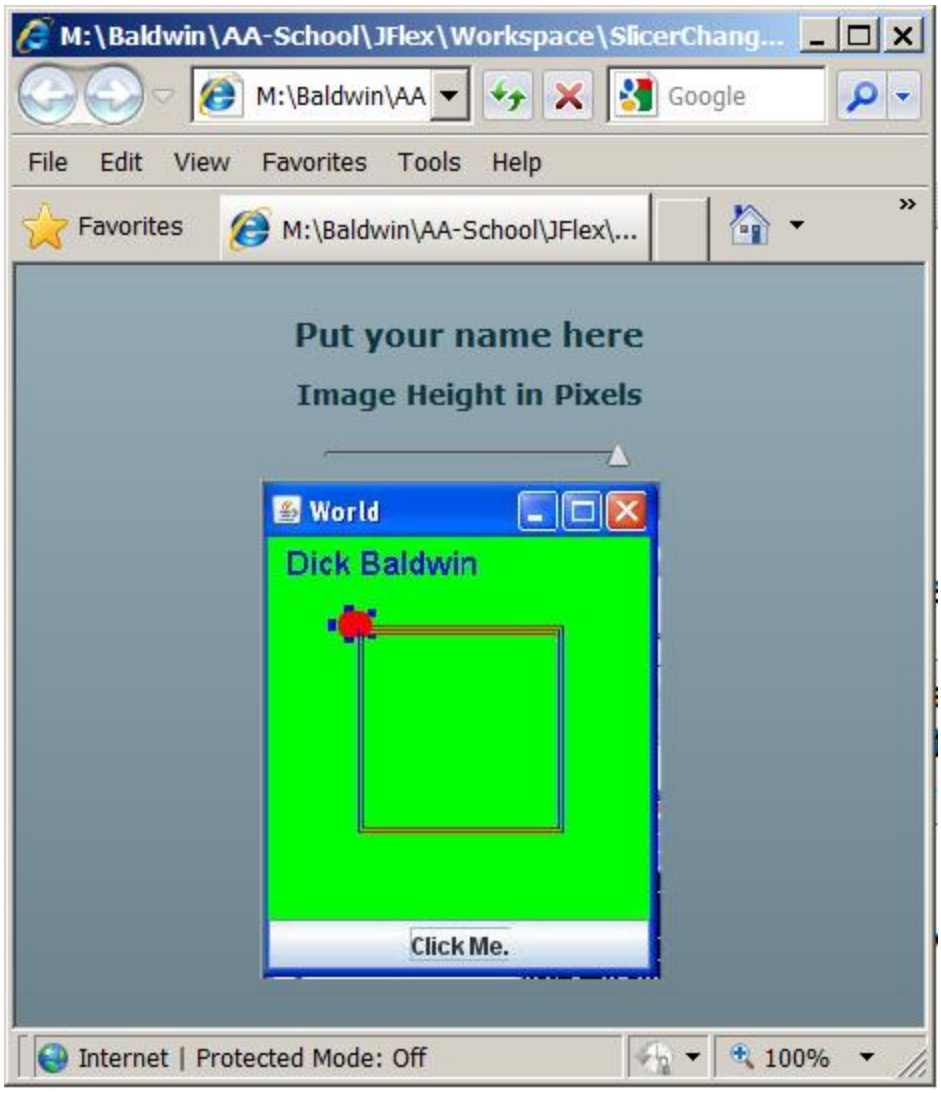
I will explain the Flex 3 project named **SliderChangeEvent01** in detail. Then I will explain the differences between that project and the hybrid project named **SliderChangeEvent02** . Finally, I will explain the differences between that project and the project named **SliderChangeEvent03** .

SliderChangeEvent01 output image at startup

The project named **SliderChangeEvent01** starts running in Flash Player with the image shown in Figure 1 appearing in the browser.

Browser image at startup for the Flex 3 project.

Figure



Browser image at startup for the Flex 3 project.

The image that you see in Figure 1 consists of two Flex 3 **Label** controls, one Flex 3 **HSlider** control, and one Flex 3 **Image** control arranged vertically and centered in the browser window.

The Application container

All XML documents must have a root element. The root of a Flex 3 application is a container element that is often called the **Application**

container. *(You can learn all about the **Application** container class at Adobe Flex 3.5 Language Reference .)*

Briefly, the **Application** container, *(which corresponds to the root element in the Flex XML code)* , holds all other containers and components.

Vertical layout

By default, the Flex 3 **Application** container lays out all of its children vertically as shown in Figure 1. *(As you will see later, this is not the case for the Flex 4 **Application** container.)* The default vertical layout occurs when the **layout** attribute is not specified as is the case in this application. According to the Adobe Flex 3.5 Language Reference , the **layout** property:

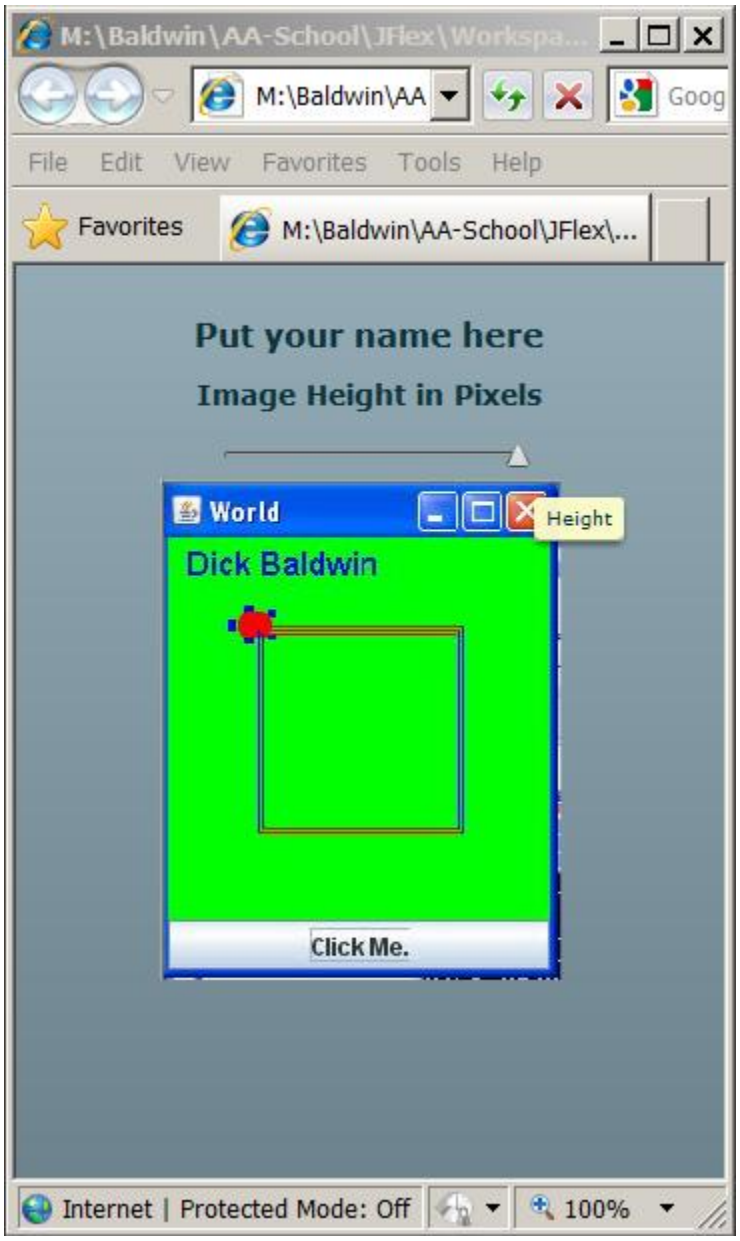
> *"Specifies the layout mechanism used for this application. Applications can use "vertical", "horizontal", or "absolute" positioning. Vertical positioning lays out each child component vertically from the top of the application to the bottom in the specified order. Horizontal positioning lays out each child component horizontally from the left of the application to the right in the specified order. Absolute positioning does no automatic layout and requires you to explicitly define the location of each child component. The default value is vertical."*

A toolTip on the slider

If you point to the slider with your mouse, a tool tip showing the word *Height* will appear as shown in Figure 2.

A toolTip on the slider.

Figure



A toolTip on the slider.

The slider's *thumb*

The little triangle that you see on the slider in these images is often referred to as the slider's *thumb* .

As you will see later, the position of the thumb is intended to represent the height of the image below the slider. The left end of the slider represents a height of 100 pixels and the right end represents a height of 250 pixels *(which just happens to be the actual height of the raw image)* .

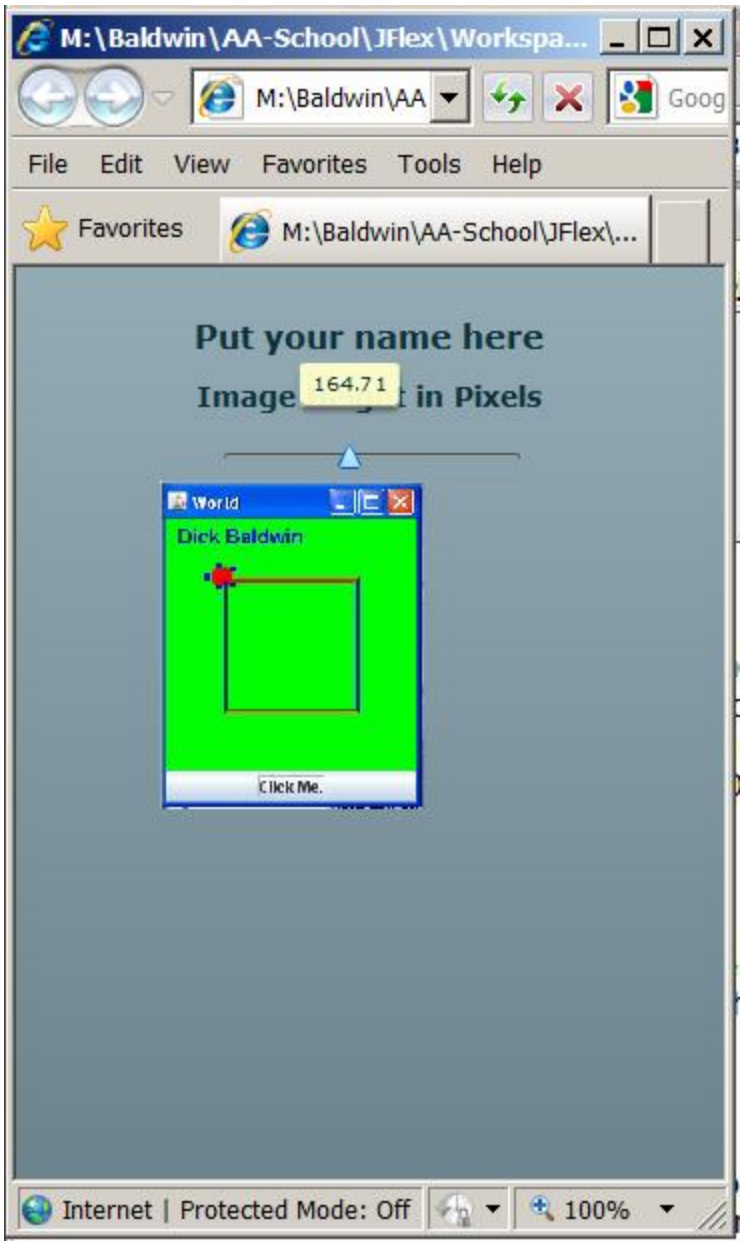
Changing the height of the image

If you grab the thumb with the mouse and move it to the left or the right, two obvious visual effects occur. The first is that the value represented by the current position of the thumb is displayed above the thumb as shown in Figure 3.

> *(As you will see later, the value is also displayed in a Flex 4 application, but by default the appearance is white numerals on a black background.)*

Changing the height of the image.

Figure



Changing the height of the image.

The second visual effect

The second visual effect of moving the thumb is that the height of the image changes to the value represented by the position of the thumb on the slider.

> *(An **Image** object has a property named **maintainAspectRatio** . By default, the value of this property is true. Therefore, when the height is changed, the width changes in a proportional manner.)*

Note that the upper-left corner of the image remains anchored to the same point as the height of the image changes as shown in Figure 3.

Discussion and sample code

The Flex 3 project named SliderChangeEvent 01

Creating the layout

Once you create your new Flex 3 Project, there are at least three ways that you can create your layout using Flex Builder 3:

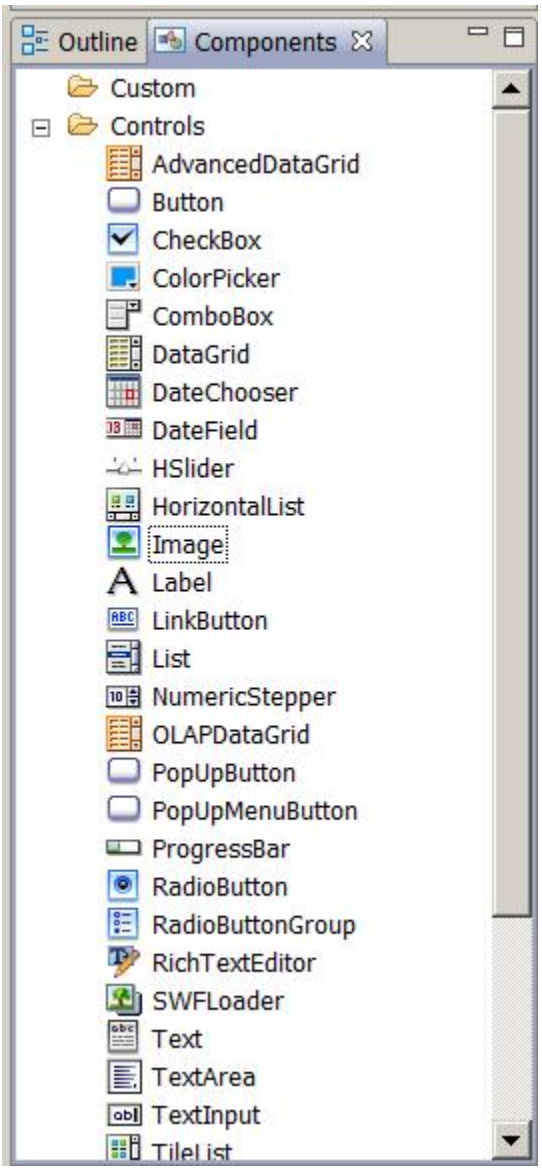
1. Select the **Design** tab in the upper-middle panel of the IDE *(see Figure 5)* and drag your containers, controls, and other components from the **Components** tab onto your design window.
2. Select the **Source** tab in Figure 5 and write the raw XML code that defines your layout.
3. A combination of 1 and 2 above

Expose the components tab

When you select the **Design** tab in the upper-middle window of the IDE, the lower-left window changes to the appearance shown in Figure 4 with the Flex 3 *(mx)* **Components** tab exposed.

Flex Builder 3 Components tab exposed.

Figure



Flex Builder 3 Components tab exposed.

> *The list of available components that you see in Figure 4 also appears when you create a new project in Flash Builder 4 and specify the use of the Flex 3 compiler.*

A list of available components

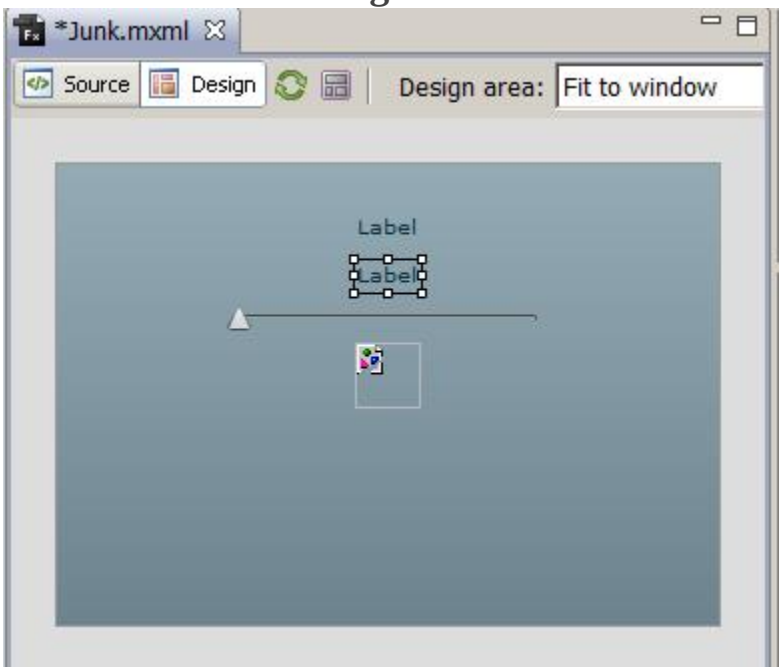
Although they aren't all shown in Figure 4 due to space limitations, the Flex Builder 3 **Components** tab lists all of the components that you can use in your Flex application grouped into the following categories:

- Custom
- Controls
- Layout
- Navigators
- Charts

Expose the design window

Selecting the **Design** tab mentioned above also exposes the Flex Builder 3 design window shown in Figure 5.
Drag controls onto the Flex Builder 3 Design tab.

Figure



Drag controls onto the Flex Builder 3 Design tab.

> *A similar design window is exposed when you create a new project in Flash Builder 4 specifying the Flex 3 compiler and then select the **Design** tab. The purpose is the same but some of the items at the top of the design window in Flash Builder 4 are different.*

Drag components onto the Design tab

You can drag components from the **Components** tab shown in Figure 4 onto the **Design** tab shown in Figure 5 to create your layout in the Flex Builder 3 design mode or in the Flash Builder 4 design mode. As you do that, the corresponding XML code is automatically generated for you.

For example, Figure 5 shows the results of dragging two **Label** controls, one **HSlider** control, and one **Image** control from the **Components** tab of Figure 4 to the **Design** tab of Figure 5. *(No attempt has been made to set property values on any of the controls shown in Figure 5.)* .

XML code before setting properties

If you select the **Source** tab at this point, you will see the XML code shown in Listing 1.

Example:
XML code before setting properties for SliderChangeEvent01.

```
<?xml version="1.0" encoding="utf-8"?>

<mx:Application
xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Label text="Label"/>
    <mx:Label text="Label"/>
    <mx:HSlider/>
    <mx:Image/>
```

```
</mx:Application>
```

Compile and run

As you can see, the XML code in Listing 1 is pretty sparse. You could compile and run the application at this point. All you would see would be two labels each containing the text **Label** and a slider covering the default numeric range from 0 to 10.

Put some meat on the bones

We will need to put some meat on the bones of this skeleton mxml code in order to create our Flex application. We can accomplish that by setting attribute values that correspond to properties of the controls.

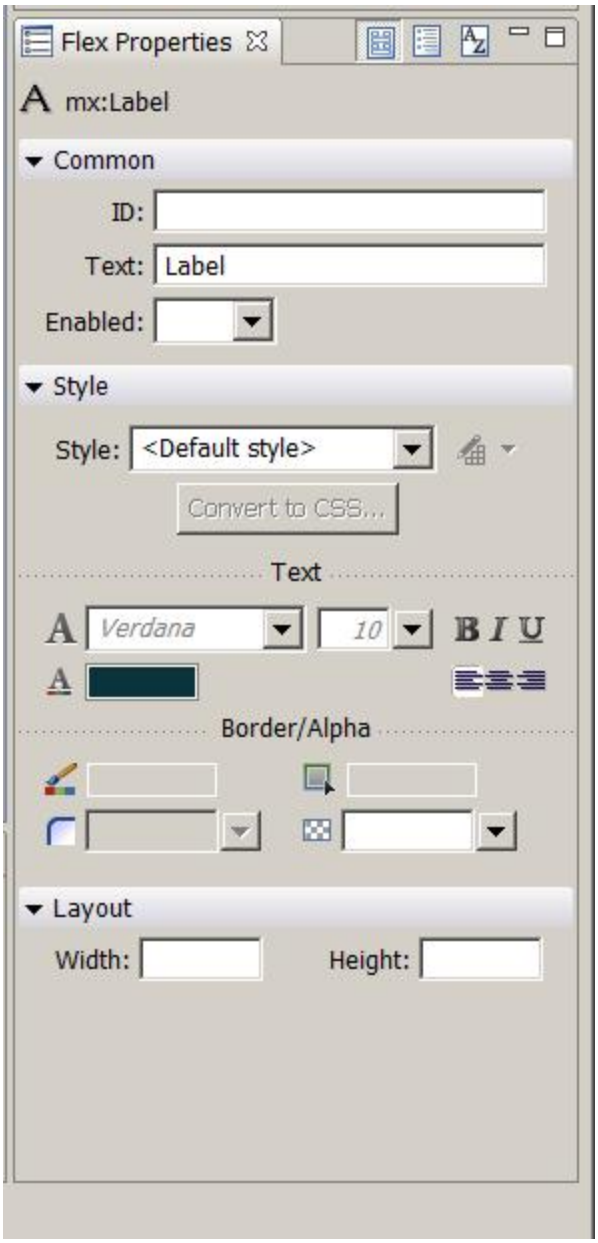
Setting attribute values

Once again, we have three choices:

1. Go hardcore and edit the XML code shown in Listing 1 to add the necessary attributes.
2. Stay in **Design** mode, select each component in the **Design** tab, and use the **Flex Properties** tab shown in Figure 6 to set the properties on that component.
3. A combination of 1 and 2 above.

The Flex Builder 3 Properties tab exposed.

Figure



The Flex Builder 3 Properties tab exposed.

The Flex Properties tab

When you select the **Design** tab shown in Figure 5, the **Flex Properties** tab shown in Figure 6 appears in the bottom-right of the IDE.

The appearance of the **Flex Properties** tab depends on which component is selected in the **Design** tab. Figure 5 shows one of the **Label** controls selected, and Figure 6 shows the **Flex Properties** tab corresponding to a **Label** control.

> *You will see a very similar properties tab if you create a new Flash Builder 4 project and specify use of the Flex 3 compiler. Some of the items are in different locations than Figure 6 but it appears that the Flash Builder 4 properties tab has the same items for a Flex 3 project.*

A variety of user input controls

The **Flex Properties** tab contains a variety of user input controls that allow you to specify values for many of the commonly used properties that are allowed for the selected component.

Note, however, that the documentation for the **Label** control lists many properties that are not supported by the **Flex Properties** tab shown in Figure 6. You can increase the number of properties shown in the tab by selecting one of the controls at the top of the tab that converts the display into an alphabetical list of properties. However, even this doesn't seem to show all of the properties defined by and inherited by some components.

If you need to set properties that are not supported by the **Flex Properties** tab, you probably have no choice but to select the **Source** tab shown in Figure 5 and write mxml code for those properties.

Will explain the code in fragments

I will explain the code for this Flex application in fragments. A complete listing of the application is provided in Listing 6 near the end of the lesson.

Beginning of XML code for SliderChangeEvent01

The primary purpose of this application is to illustrate the use of inline event handling for Flex 3 slider *change* events.

The application begins in Listing 2 which shows the beginning of the **Application** element and the two complete **Label** elements shown at the top of Figure 1.

Example:
Beginning of XML code for SliderChangeEvent01.

```
<?xml version="1.0" encoding="utf-8"?>

<mx:Application
xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Label text="Put your name here"
fontSize="16"
        fontWeight="bold"/>

    <mx:Label text="Image Height in Pixels"
        fontWeight="bold" fontSize="14"/>
```

Make it easy - drag and drop

The two **Label** elements were created by dragging **Label** controls from the **Components** tab shown in Figure 4 onto the **Design** tab shown in Figure 5. Then the attribute values were set using the **Flex Properties** tab shown in Figure 6.

The attributes shown in Listing 2 represent common properties of a text label and shouldn't require further explanation.

Create and condition the slider

Listing 3 adds a horizontal slider *(HSlider)* control to the application and sets the attributes that control both its appearance and its behavior.

Example:
Create and condition the slider.

```
    <mx:HSlider minimum="100" maximum="250"
value="250"
        toolTip="Height"

change="myimg.height=event.currentTarget.value"
        liveDragging="true" />
```

The slider is a little more complicated than a label and deserves a more thorough explanation.

The numeric properties

Recall that a slider represents a range of numeric values. The position of the thumb at any instant in time selects a value from that range. The following three attributes shown in Listing 3 deal with the slider and its numeric range:

- **minimum** - the numeric value represented by the left end of a horizontal slider.
- **maximum** - the value represented by the right end of a horizontal slider.
- **value** - the value that specifies the initial position of the thumb when the slider is constructed and first presented in the application's window.

The toolTip property

As you have probably already guessed, the value of the **toolTip** property specifies the text that appears in the tool tip when it is visible as shown in

Figure 2.

The change property

This is where thing get a little more interesting. As the user moves the thumb to the left or right, the slider fires a **continuous** stream of **change** events. You might think of this as the slider yelling out *"Hey, the position of my thumb has been changed."* over and over as the thumb is being moved. *(Also see the discussion of the liveDragging property later.)*

An *event handler*

The value that is assigned to the **change** attribute in Listing 3 is often referred to as an *event handler* . This value specifies what the application will do each time the slider fires a **change** event.

Three ways to handle events in Flex

There are at least three ways to handle event notifications in Flex:

- Registering an event handler in mxml
- Creating an **inline event handler** in the mxml definition
- Registering an event listener through ActionScript

The XML code in Listing 3 uses the inline approach.

The inline approach

The advantage of using the inline approach, *(at least insofar as my **Introduction to XML** , students are concerned)* , is that it doesn't require you to create a **Script** element within the mxml or to create a separate ActionScript file.

Handling the slider's change event

Now consider the code that begins with the word **change** in Listing 3. The code within the quotation marks can be a little hard to explain, but I will give it a try. *(The code in quotation marks is actually an ActionScript code fragment.)*

Think of it this way

There is a Flex/ActionScript class named **Event** . The reference to **event** in Listing 3 is a reference to an object of the **Event** class that comes into being each time the slider fires a **change** event **.**

The **Event** object encapsulates a property named **currentTarget** , which is described in the Flex 3 documentation as follows:

> *"The object that is actively processing the Event object with an event listener. For example, if a user clicks an OK button, the current target could be the node containing that button or one of its ancestors that has registered an event listener for that event."*

The currentTarget is the slider

In this application, the value of **currentTarget** points to the slider which is firing **change** events as the user moves the thumb.

The *value* property of an HSlider object

The slider is an object of the **HSlider** class, which has a property named **value** . The **value** property contains the current position of the thumb and is a number between the *minimum* and *maximum* property values.

Get the current value of the thumb

Therefore, each time the slider fires a **change** event, the code on the right side of the assignment operator within the highlighted quotation marks in Listing 3 gets the numeric value that indicates the current position of the thumb.

Cause the image to be resized

This value is assigned to the **height** property of the image, causing the overall size of the image to be adjusted, if necessary, to match the current position of the slider's thumb. *(I could go into more detail as to the sequence of events that causes the size of the image to change, but I will leave that as an exercise for the student.)*

The liveDragging property

That leaves one more attribute or property for us to discuss in Listing 3: **liveDragging** . This one is much easier to understand.

The Flex 3 documentation has this to say about the **liveDragging** property:

> *"Specifies whether live dragging is enabled for the slider. If false, Flex sets the value and values properties and dispatches the change event when the user stops dragging the slider thumb. If true, Flex sets the value and values properties and dispatches the change event continuously as the user moves the thumb. The default value is false."*

If liveDragging is false...

If you were to modify the code in Listing 3 to cause the value of the **liveDragging** property to be false *(or simply not set the attribute value to true)* , the slider would only fire change events each time the thumb stops moving *(as opposed to firing a stream of events while the thumb is moving)* . This, in turn, would cause the size of the image to change only when the thumb stops moving instead of changing continuously while the thumb is moving.

An Image control

The Flex 3 documentation tells us:

>

> *The Image control lets you import JPEG, PNG, GIF, and SWF files at runtime. You can also embed any of these files and SVG files at compile time by using @Embed(source='filename').*

The primary output that is produced by compiling a Flex application is an swf file that can be executed in Flash Player.

The documentation goes on to explain that by using **@Embed** , you can cause resources such as images to be embedded in the swf file.

The advantage to embedding is that embedding the resource eliminates the requirement to distribute the resource files along with the swf files. The disadvantage is that it causes the swf file to be larger.

Import an image

Listing 4 imports an image from the file named **myimage.jpg** that is located in the **src** folder of the project tree. This image is embedded in the swf file when the Flex application is compiled.

Example:
Import an image.

```
    <mx:Image id="myimg"
source="@Embed('myimage.jpg')"
        height="250">
    </mx:Image>

</mx:Application>
```

The *id* property

Setting the **id** property on the image to **myimg** makes it possible to refer to the image in the change-event code in Listing 3.

> *(Note that there is no requirement to set the value of the **id** property to be the same as the name of the image file as was done in Listing 4.)*

The *height* property

Setting the **height** property of the image to 250 pixels in Listing 4 causes the image height to be 250 pixels when it is first displayed as shown in Figure 1.

The end of the application

Listing 4 contains the closing tag for the **Application** element signaling the end of the Flex 3 application named **SliderChangeEvent01** .

The hybrid Flex3-4 project named SliderChangeEvent02

The mxml project code

The mxml code for this project is shown in its entirety in Listing 7.

If you examine this code you will see that:

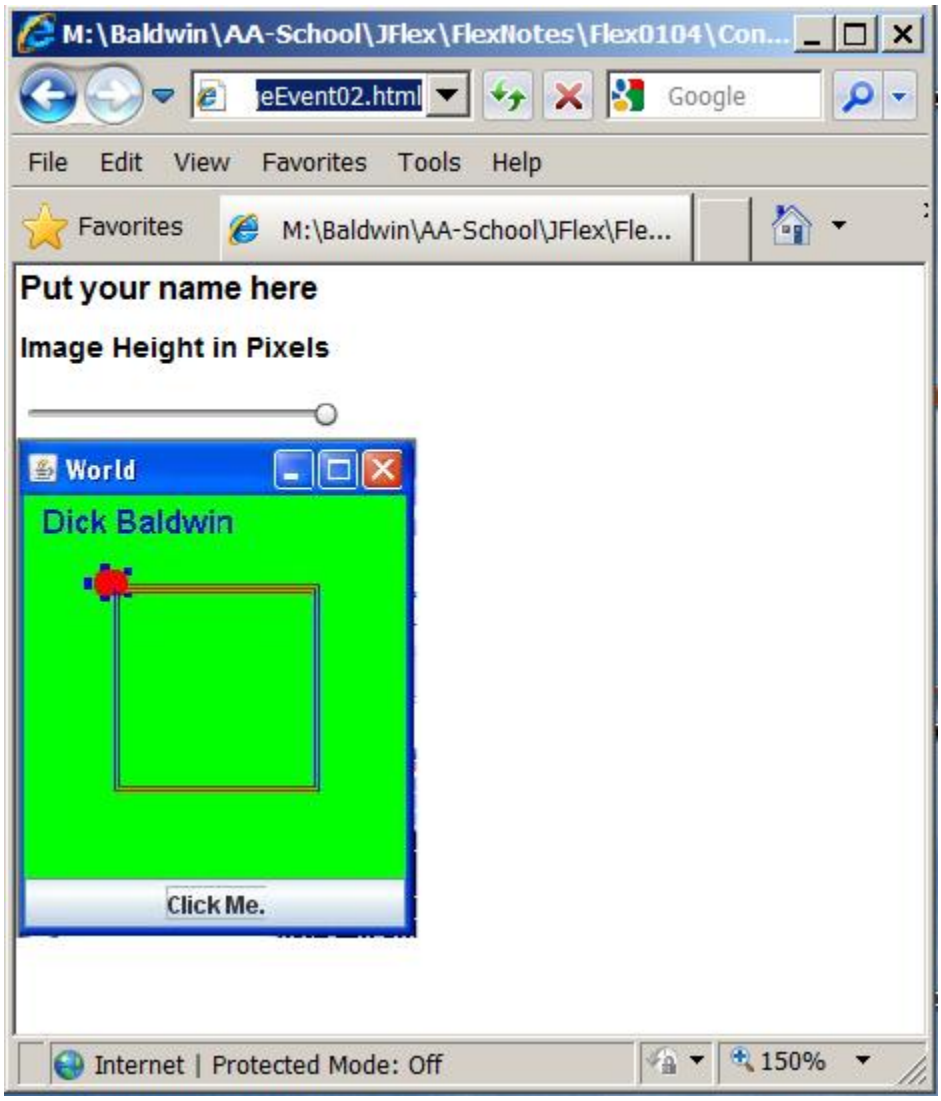
- It uses a Flex 4 spark **s:Application** element instead of a Flex 3 **mx:Application** element.
- It declares the standard set of Flex 4 namespaces.
- It uses a spark **s:VGroup** element as the container for the following Flex 3 components. *(Note that the Flex 3 project in Listing 6 doesn't require another container in addition to the **mx:Application** container.)* :

- **mx:Label**
- **mx:Label**
- **mx:HSlider**
- **mx:Image**

Otherwise, the mxml code for this project is the same as the code for the Flex 3 project shown in Listing 6. The mixture of spark and mx components causes this to be a hybrid Flex 3-4 project.

Visual appearance of the project

If you run the online version of the project named **SliderChangeEvent02** , you should see an initial screen display similar to Figure 7.
Browser image at startup for the hybrid project.
Figure



Browser image at startup for the hybrid project.

Compare with the Flex 3 project

By comparing this screen output for the hybrid project with the screen output for the Flex 3 project in Figure 1, you can immediately spot several significant differences:

- The background is white instead of gray.
- The labels, the slider, and the image are not centered horizontally in the browser window.

- The appearance of the thumb on the slider is a circle instead of a triangle.

Behavior of the project

If you move the slider with the mouse, you will see that the behavior is essentially the same as the Flex 3 version of the project, including the display of a tool tip as shown in Figure 2 and the display of the slider value as shown in Figure 3.

The spark s:Application element

The Flex 4 spark **s:Application** element differs from the Flex 3 **mx:Application** element in several ways including the following:

- **Default layout:** Unlike the **mx:Application** element, the **s:Application** element does not have a default vertical layout. By default, all components placed in the **s:application** element are placed in the upper-left corner. *(The **s:VGroup** container element was used in Listing 7 to resolve this issue.)*
- **Default background color:** The default background color of the **s:Application** element is white, whereas the default background color of the **mx:Application** element is gray.
- **Horizontal positioning:** Unlike the mx:Application element which centers it components horizontally by default, the default position of components placed in the **s:Application** element is the upper-left corner.

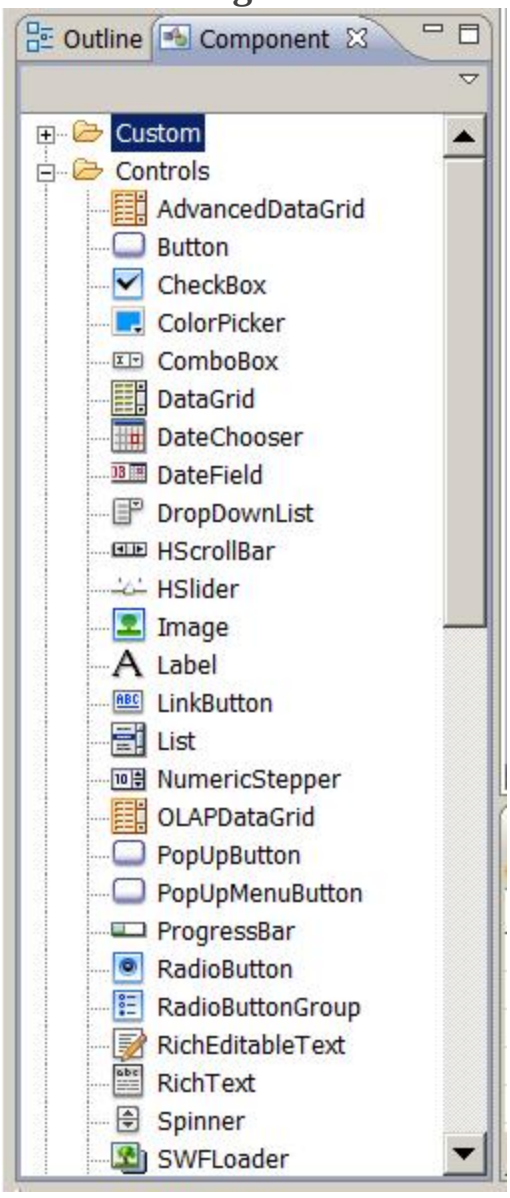
The spark s:VGroup element

As shown in Listing 7, the **s:VGroup** element can be used to arrange the components in a vertical sequence from top to bottom. However, placing components in an **s:VGroup** element does not cause them to be centered horizontally. Instead, by default, the components end up on the left side of the container as shown in Figure 7. If you want the components to be centered, you must write additional code to cause that to happen.

The Flash Builder 4 Components tab

When you create a new Flex project in Flash Builder 4, if you specify the use of the Flex 3 compiler, the **Components** tab in the resulting IDE will look like Figure 4. However, if you specify the Flex 4 compiler when you create the new project, the **Components** tab will look like Figure 8.
Flex Builder 4 Components tab exposed.

Figure



Flex Builder 4
Components tab exposed.

Numerous differences

If you compare Figure 8 with Figure 4, you will see numerous differences between the two lists. Some of the names are the same and some of the names are different. Even though some of the names are the same, most of the components that you see in Listing 8 are Flex 4 spark components and the components that you see in Figure 4 are Flex 3 mx components and they are represented by different classes in the class library.

> *However, as you will see later, the **Image** component in the Flex 4 Component list is actually a Flex 3 mx component. That may also be the case for some of the other components as well.*

Same name doesn't guarantee same appearance or same behavior

While the appearance and behavior of a Flex 4 spark component may be the same as the appearance and behavior of a Flex 3 mx component with the same name, there is no guarantee that will be the case. They are entirely different components and the only way you can be sure is to study the documentation.

> *As you saw earlier, the appearance of an **mx:HSlider** used inside an **s:Application** element is different from an **mx:HSlider** used inside an **mx:Application** element. Therefore, if the appearance and behavior of the components in your project are really critical, you should probably avoid mixing Flex 3 and Flex 4 components.*

No drag-and-drop support for hybrid projects

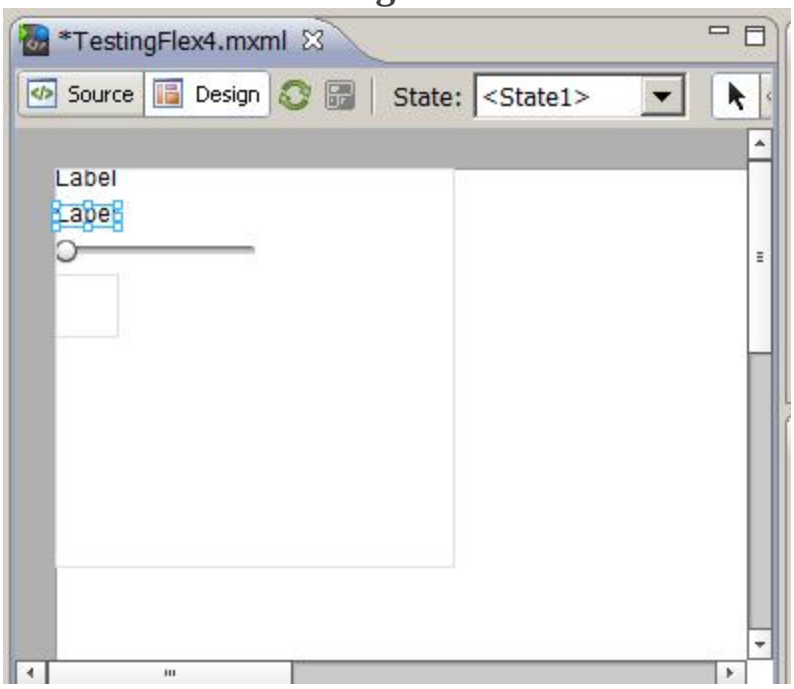
Another ramification of the fact that the components in Figure 8 are spark components is that you cannot create hybrid projects using drag-and-drop programming alone. If you drag the components in Figure 8 into the **design**

pane of Flash Builder 4, your project will be populated with Flex 4 spark components. If you want the project to be populated with Flex 3 mx components, you will have to manually edit the mxml code to accomplish that. That may be another reason to avoid hybrid projects.

Drag-and-drop results

Figure 9 shows the results of dragging one **VGroup** layout, two **Label** controls, one **HSlider** control, and one **Image** control from the Flash Builder 4 **Components** panel into the **Design** panel.
Drag controls onto the Flash Builder 4 Design tab.

Figure



Drag controls onto the Flash Builder 4 Design tab.

Compare Figure 9 with Figure 5 to see the differences in background color and layout.

Mxml code for the layout shown in Figure 9

Listing 5 shows the Flex 4 mxml code that corresponds to the layout shown in Figure 9.

Example:
Mxml code for the layout shown in Figure 9.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application
xmlns:fx="http://ns.adobe.com/mxml/2009"

xmlns:s="library://ns.adobe.com/flex/spark"

xmlns:mx="library://ns.adobe.com/flex/mx"
                minWidth="955" minHeight="600">
  <fx:Declarations>
    <!-- Place non-visual elements (e.g.,
services,
    value objects) here -->
  </fx:Declarations>
  <s:VGroup x="0" y="0" width="200" height="200">
    <s:Label text="Label"/>
    <s:Label text="Label"/>
    <s:HSlider/>
    <mx:Image/>
  </s:VGroup>
</s:Application>
```

Mostly Flex 4 spark components

The most important thing to note about Listing 5 is that the **VGroup** , **Label** , and **HSlider** components that were dragged from the **Component** tab shown in Figure 8 into the **Design** panel shown in Figure 9 are all

declared using the spark *(s)* namespace. Curiously, however, the **Image** control shows up in the code as **mx:Image** instead of **s:Image** .

The Flex 4 project named SliderChangeEvent03

Updating the mxml code in Listing 5 by applying the properties from Listing 7 to the **Label** and **HSlider** components produces the complete Flex 4 project named **SliderChangeEvent03** shown in Listing 8.

If you run the online versions of **SliderChangeEvent0** 2 and **SliderChangeEvent03** side-by-side in different browser windows, you will probably notice a few subtle differences in the look and feel of the two programs. Here are some of the differences that I have noticed:

- There is less space between the labels and the slider in the Flex 4 version.
- There is less space between the edge of the Flash window and the top and left ends of the labels and the slider in the Flex 4 version.
- The overall length of the slider is shorter in the Flex 4 version.
- The treatment of the little popup window that shows the value of the slider is different between the two. It has black letters on a cream-colored background in the hybrid version as in Figure 3, but it has white letters on a black background in the Flex 4 version.

Run the programs

I encourage you to run the online versions of the programs from this lesson. Then copy the code from Listing 6, Listing 7, and Listing 8. Use that code to create Flex projects of your own. Compile and run your projects. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Resources

I will publish a list containing links to Flex resources as a separate document. Search for Flex Resources in the Connexions search box.

Complete program listing

Complete listings of the programs discussed in this lesson are shown in Listing 6, Listing 7, and Listing 8 below.

Example:
Complete listing of SliderChangeEvent01.

```
<?xml version="1.0" encoding="utf-8"?>

<!--
SliderChangeEvent01
Illustrates the use of inline event handling for
slider
change events.
-->

<mx:Application
xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Label text="Put your name here"
fontSize="16"
        fontWeight="bold"/>

    <mx:Label text="Image Height in Pixels"
        fontWeight="bold" fontSize="14"/>

    <mx:HSlider minimum="100" maximum="250"
value="250"
        toolTip="Height"

change="myimg.height=event.currentTarget.value"
        liveDragging="true" />
```

```
    <mx:Image id="myimg"
source="@Embed('myimage.jpg')"
        height="250">
    </mx:Image>
</mx:Application>
```

Example:
Complete listing of SliderChangeEvent02.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/mx">
  <s:VGroup>
    <mx:Label
      text="Put your name here" fontSize="16"
      fontWeight="bold"/>

    <mx:Label
      text="Image Height in Pixels"
      fontWeight="bold" fontSize="14"/>

    <mx:HSlider
      minimum="100"
      maximum="250"
      value="250"
      toolTip="Height"

change="myimg.height=event.currentTarget.value"
      liveDragging="true" />

    <mx:Image
```

```
      id="myimg" source="@Embed('myimage.jpg')"
      height="250">
    </mx:Image>

  </s:VGroup>
</s:Application>
```

Example:
Complete listing of SliderChangeEvent03.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application
xmlns:fx="http://ns.adobe.com/mxml/2009"

xmlns:s="library://ns.adobe.com/flex/spark"

xmlns:mx="library://ns.adobe.com/flex/mx"
               minWidth="955" minHeight="600">

  <s:VGroup x="0" y="0" width="200" height="200">
    <s:Label
      text="Put your name here" fontSize="16"
      fontWeight="bold"/>
    <s:Label
      text="Image Height in Pixels"
      fontWeight="bold" fontSize="14"/>
    <s:HSlider
      minimum="100"
      maximum="250"
      value="250"
      toolTip="Height"

change="myimg.height=event.currentTarget.value"
      liveDragging="true" />
```

```
    <mx:Image id="myimg"
source="@Embed('myimage.jpg')"
              height="250"/>
  </s:VGroup>
</s:Application>
```

Miscellaneous

Note: Housekeeping material

- Module name: Handling Slider Change Events in Flex 3 and Flex 4
- Files:
    - Flex0104\Flex0104.htm
    - Flex0104\Connexions\FlexXhtml0104.htm

-end-

Flex Resources
The purpose of this document is to provide a list of links to online Flex and ActionScript resources to supplement the other lessons in the series.

Table of Contents

Preface

This tutorial lesson is part of a series of lessons dedicated to programming using Adobe Flex.

The purpose of this document is to provide a list of links to online Flex and ActionScript resources to supplement the other lessons in the series.

Note: The material in these lessons is based on Flex version 3 and Flex version 4.

Resources

- Baldwin's Flex programming website
- Baldwin's ActionScript programming website
- Adobe Flex Home
- Download free open-source Adobe Flex 3.5 SDK
    - Adobe Flex SDK Installation and Release Notes
    - Application Deployment
- Download free open-source Adobe Flex 4 SDK
- Download free FlashDevelop IDE

    - Getting Started with FlashDevelop

- Download Adobe Flash Builder 4 Standard for students
- Download Adobe Flash Player
- Download Adobe Flash Player Uninstallers
- Download Adobe Air
- Download various Adobe products
- Flex Developer Center
- Flex in a Week video training
- Adobe Flex Builder 3 - Getting Started
- Getting Started with Flex 3 - online O'Reilly book by Jack Herrington and Emily Kim
- Adobe Flex 3 Help

    - Adobe Flex 3.5 Language Reference
    - Building and Deploying Flex 3 Applications
    - Programming ActionScript 3.0
    - ActionScript language and syntax

- Adobe Flex 4 reference material
- Using Flash Builder 4
- Flex.org
- Wikipedia on MXML
- ActionScript 3 guides, tutorials, and samples
- ActionScript.org
- ActionScript 3: The Language of Flex
- ActionScript Custom Components
- ActionScript language and syntax
- Comparing, including, and importing ActionScript code
- Programming ActionScript 3.0
- Getting Started with ActionScript 3.0
- Modular applications overview
- ActionScript 3 Language Specification
- Beginners Guide to Getting Started with AS3 *(Running the compiler from the command line.)*
- Tips for learning ActionScript 3.0
- ActionScript Technology Center

- Adobe Flash Platform
- Adobe Flash Player
- Adobe Air
- ActionScript language references
- Class property attributes
- Embedding Resources with AS3

Miscellaneous

This section contains a variety of miscellaneous materials.

Note: Housekeeping material

- Module name: Flex Resources
- Files:
    - Flex9999\Connexions\FlexXhtml9999.htm

-end-